Noname manuscript No.
(will be inserted by the editor)

Making Data Visualization More Efficient and Effective: A Survey

Xuedi Qin[†] · **Yuyu Luo**[†] · **Nan Tang**[‡] · **Guoliang Li**[†]

Abstract Data visualization is crucial in today's data driven business world, which has been widely used for helping decision making that is closely related to major revenues of many industrial companies. However, due to the high demand of data processing *w.r.t.* the volume, velocity and veracity of data, there is an emerging need for database experts to help for efficient and effective data visualization. In response to this demand, this article surveys techniques that make data visualization more efficient and effective. (1) *Visualization specifications* define how the users can specify their requirements for generating visualizations. (2) *Efficient approaches for data visualization* process the data and a given visualization specification, which then produce visualizations with the primary target to be efficient and scalable at an interactive speed. (3) *Data visualization recommendation* is to auto-complete an incomplete specification, or to discover more interesting visualizations based on a reference visualization.

1 Introduction

Data visualization, which transforms abstract data into physical visions (for example, length, position, shape, color, and so on), is a powerful means to present compelling stories of data to humans who are more visually oriented. Nowadays, all organizations have more data than ever at their disposal. Consequently, more and more organizations use data and advanced analytics to inform strategic and operational decisions. Data visualization is a natural fit for both giving a good overview of massive data, and making it easier to interpret the results of data analytics to data scientists.

The Blossom of Data Visualization. Undoubtedly, data visualization has made great strides in many fields, contributed by multiple communities.

The *computer graphics* community has significantly advanced the technology of rendering beautiful yet self-interpretable visualizations using e.g., D3 [87].

The *visualization community* makes it easy for users to specify and interact with visualizations, such as D3 [87], Vega-Lite [109], VizQL [54], Tableau [11] and Microsoft Power BI [9].

The *database community* has significantly improved the user experience of seeing and interacting with data visualization in real-time, even for big data (e.g., for millions or billions of records). For example, Hyper DB [5, 92, 93] is the back-end engine to power up Tableau [11], and the Falcon project (available at GitHub `https://github.com/uwdata/falcon`) makes D3 [87] highly scalable supported by Apache Spark.

In addition, data visualization has also been extensively used in many database-related applications, such as Excel [7], Google Sheets [4], Oracle Data Visualization Desktop [8], IBM DB2 [6], Amazon Quicksight [1], Microsoft Power BI [9], and many others.

The Pipeline of Data Visualization. A typical *iterative data visualization pipeline*[1] is shown in Figure 1.

Xuedi Qin
E-mail: qxd17@mails.tsinghua.edu.cn

Yuyu Luo
E-mail: luoyy18@mails.tsinghua.edu.cn

Nan Tang
E-mail: ntang@hbku.edu.qa

Guoliang Li (Corresponding author)
E-mail: liguoliang@tsinghua.edu.cn

[†]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[‡]Qatar Computing Research Institute, HBKU, Doha, Qatar

[1] Note that, our pipeline and terminologies used in this paper are slightly different than those used in the visualization

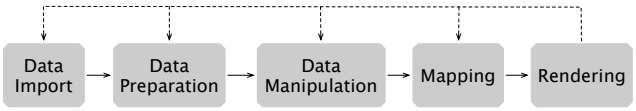


Fig. 1 The data visualization pipeline.

1. *Data import* is to retrieve the required data from a desired data source.
2. *Data preparation* is to prepare the imported data for visualization, by e.g., normalizing values, correcting erroneous entries, and interpolating missing values.
3. *Data manipulation* is to select the data to be visualized (*a.k.a.* filtering from the visualization community), and possibly with other common operations such as joining and grouping.
4. *Mapping* is to map the data obtained from the above process to geometric primitives (e.g., points and lines), together with their attributes (e.g., color, position and size).
5. *Rendering* is to transform the above geometric data into a visual representation.

Based on the pipeline, we have identified three directions that make data visualization more efficient and effective, yet relevant to database researchers.

(1) ***Visualization Specifications.*** Visualization specifications provide various ways that users can specify what they want. There have been a great many studies from both visualization [12,73,87,109,140] and database community [54, 64, 81, 119] on visualization specifications. We include it in this survey for two reasons:

- *Self-completeness:* It is important for readers to know how to generate data visualizations.
- *Language design perspective:* It mainly serves the "Mapping" component of the pipeline (Figure 1), by specifying how to map different information to visual elements. However, it has some overlap with the "Data Manipulation" component, e.g., grouping and ordering operations can be specified in either step, which triggers a design choice problem between database languages (such as SQL) and visualization languages (see Section 2 for more details).

(2) ***Efficient Approaches for Data Visualization.*** In order to effectively involve users in the iterative pipeline, the process of creating data visualizations must be efficient and scalable, especially for the two components, "Data Manipulation" and "Mapping". Many researchers have tried both interfacing with powerful and mature data processing engines (such as translating visualization queries to SQL queries to be evaluated over RDBMSs [81, 98, 125, 132, 133]), and customizing existing systems for data visualization tasks (such as HyperDB [5, 92, 93] for Tableau). There are also approximate solutions [40, 88] and progressive solutions [48, 71, 101] to cope with big data, in order to provide real-time response. Both visualization [48, 77, 87, 136, 139] and database communities [40, 42, 82, 132, 149] have signification contributions on efficient visualization.

(3) ***Data Visualization Recommendation.*** Precisely specifying a visualization is hard, even for experts, simply because the understanding of *what data to visualize*, *which story to tell*, and *how to visualize* is a trial and error exercise [81, 90, 118, 132]. Hence, it is important that the visualization system can smartly guide users by providing recommendations. Several systems [82, 99, 118, 119] allow users to provide an ambiguous specification, and the system will either automatically complete the visualizations, or provide recommendations. The works [84, 113, 125, 135, 145] from visualization community and [65, 81, 119, 132] from database community tackle the problem of visualization recommendation from various angles.

Related Surveys. Most existing surveys on visualization focus on a specific topic, such as graph visualization [21, 57, 134], linked data visualization [25, 35, 85], ontology visualization [67], high-dimensional data visualization [76], temporal data visualization [143]. We survey techniques from a different perspective.

For *visualization specifications*, Mei et al. [86] give a survey about classification, data source, presentation medium etc. of visualization languages. We survey visualization languages from the stack perspective, and emphasize how these languages are used from a practical perspective. There have also been some surveys [38, 50] on exploratory data analysis tools, which is complementary to our *interactive data visualization* – we have added a discussion in the corresponding section.

For *efficient approaches for data visualization*, Keim et al. [68] consider how to integrate databases, data visualization, and data analysis so a user can easily work in one system, but without a discussion for efficiency. Idreos et al. [62] surveyed the techniques which aim to improve efficiency in the data exploration cycles, but we focus on techniques about how to construct visualizations efficiently. Bikakis [23] gives an overview of current systems and techniques for big data visualization, but with a less detailed discussion.

For *data visualization recommendation*, although there have been many works [14, 26, 29, 115] about recommendation systems and works about recommendation for different tasks, e.g., QOS-aware web ser-

community. Please refer to `https://infovis-wiki.net/wiki/Visualization_Pipeline` for more details.

vices [32], social software [53], E-commerce [138], there is no survey about data visualization recommendation, where we survey how different systems recommend insightful visualizations for users automatically.

2 Visualization Specifications

2.1 The Specification of Data Visualizations

Generally speaking, data visualization languages consist of three parts: data, marks (or visual cues), and the mapping between them.

- **Data**
  - *Records*: the data that needs to be visualized.
  - *Transformation*: the operations – such as group, bin, filter, and sort – that are used to transform the specified data records.
- **Marks** (or visual cues)
  - *Type:* the visual representation for data records, such as bar, line or point.
  - *Size:* the width, height of the visualization.
  - *Legend:* the legend information.
  - *Miscellaneous:* other properties, such as the width and color of a bar.
- **Mapping**: maps data to corresponding marks.

GUI-based visual operations are typically translated into data visualization languages.

2.2 A Categorization of Data Visualization Languages

A commonly used strategy to categorize data visualization languages is based on their expressiveness, as shown in the left side of Figure 2. Apparently, the lower level of a language, the more expressive it is. Higher level languages encapsulate some low-level details by providing sensible defaults and adding more constraints (e.g., Excel [7] provides templates for supported visualizations). Another dimension to understand different levels of visualization specification languages is through their *accessibility* (or easy-to-use): the higher level the language, the easier to use, as also shown in Figure 2.

Low-level Languages. We refer to low-level languages as those that the users need to specify all mapping elements [3, 12, 27, 55, 87, 110].

Prefuse [55] and Flare [3] are Java based visualization libraries; they encapsulate visual items as a Java class, which have many visual attributes, and the languages map data to these visual attributes by setting predefined functions. Protovis [27] is a declarative JavaScript based graphical toolkit; it uses simple

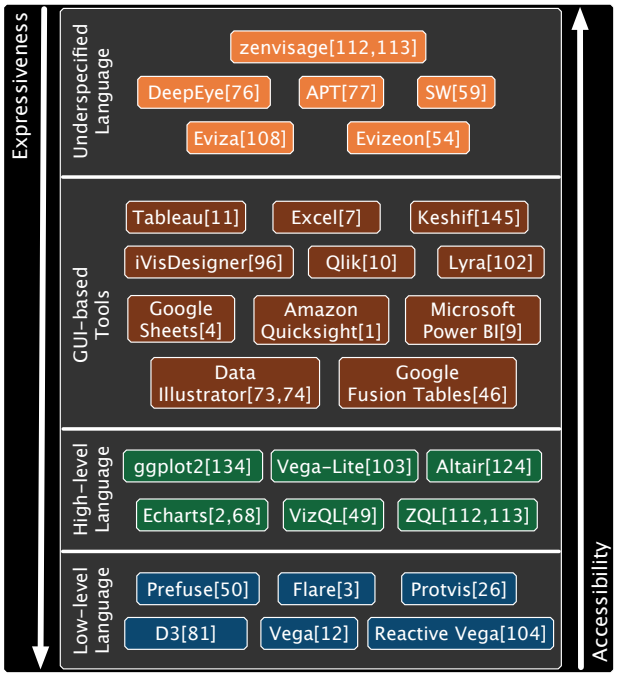


Fig. 2 An overview of data visualization specifications. Data visualization specifications are classified to four types: Low-level Language, High-level Language, GUI-based Tools and Underspecified Language. The higher level the data visualization specification is, the easier it is to use and the less expressive it is.

graphical marks (bar, area, line, etc.) with specified visual attributes. D3 [87] is a development of Protovis and is more effective in dealing with users' interaction (e.g., brushing and linking [70]).Vega [12] and Reactive Vega [110] are similar to Protovis and D3, but they provide declarative composable interaction grammars.

High-level Languages. High-level languages [2,54,73, 109,118,119,131,141,142] encapsulate the details of visualization construction, such as the mapping function, as well as some properties for marks such as canvas size, legend, and other properties.

ggplot2 [141] is built on top of Wilkinson's work in 2005 "*The Grammar of Graphics* [142]"; it is a layered grammar of graphics embedded in R language. Vega-Lite [109] is a higher development of Vega and Reactive Vega; it also supports composable interaction design but provides concise grammars. Recently, Altair [131] made Vega-Lite available to the Python community. Echarts [2, 73] is a latest development in declarative visualization languages designed to support quick visualization creation for non-programmers. VizQL [54] develops from the Polaris system [125], and is the visualization specification language of Tableau. ZQL [118, 119] of Zenvisage [118, 119] employs a tab-

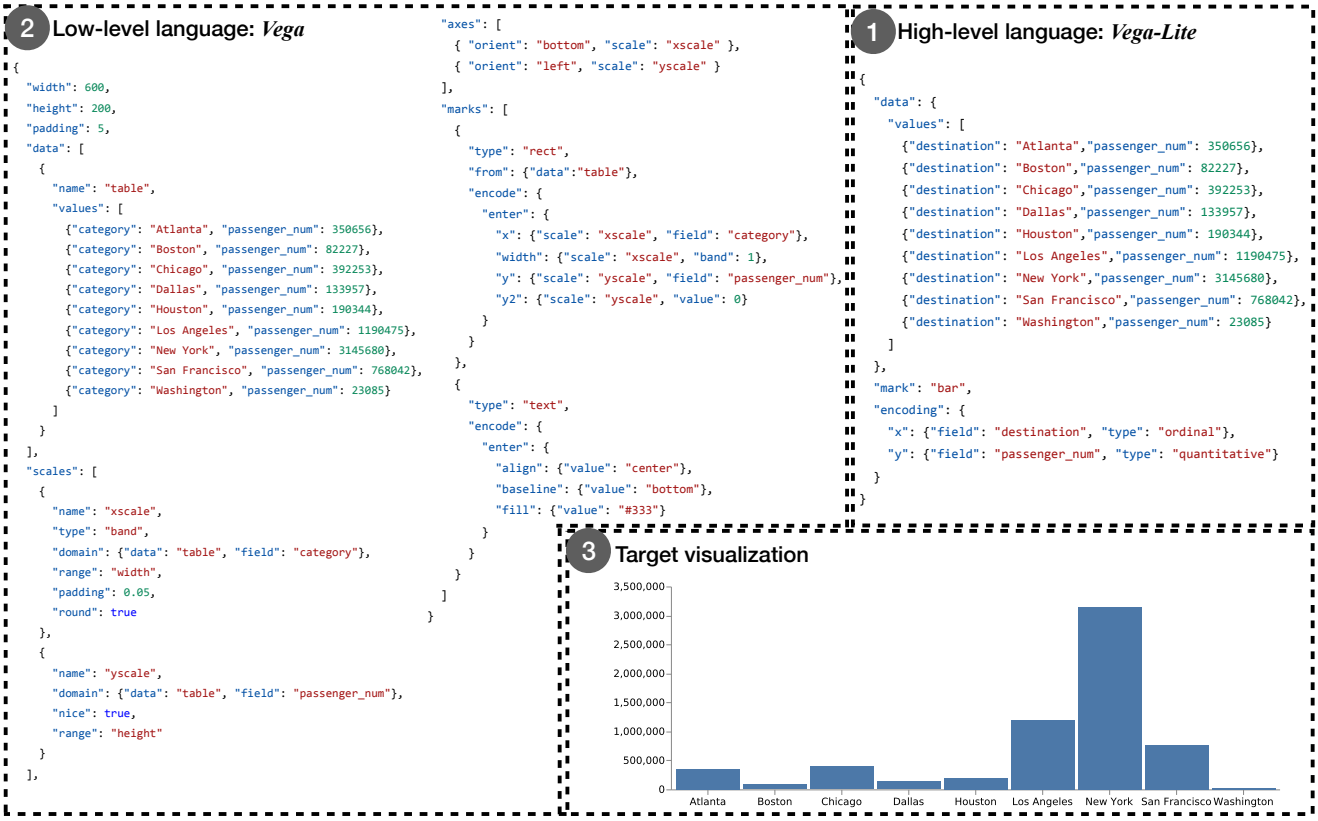


Fig. 3 Example of low- and high-level visualization languages. The target visualization (③) is a bar chart showing the passenger_num of different destinations. And we can use both low- (②) and high-level (①) visualization language to specify ③.

A. scheduled	B. carrier	C. destination	D. departure delay (min)	E. arrival delay (min)	F. passengers
01-Jan 00:04	AA	New York	-5	2	173
01-Jan 06:43	MQ	Atlanta	9	2	132
01-Jan 09:30	EV	Chicago	13	17	127
01-Jan 00:04	AA	Boston	22	10	141
01-Jan 00:04	MQ	New York	19	13	232
01-Jan 00:04	UA	Los Angeles	0	-2	119

Table 1 An excerpt of flight delay statistics of Chicago O'Hare International (Jan - Dec, 2015), where *scheduled* is the scheduled time to take off, *carrier*, *destination*, *departure delay (min)*, *arrival delay (min)*, *passengers* are the carrier, destination, departure delay, arrival delay, passenger number of the flight respectively.

ular structure language – each row in the table is a visualization specification.

Now let us show the difference between different levels of visualization languages by an example.

Example 1 Table 1 is an excerpt of flight delay statistics. And Figure 3 shows high-level (Figure 3–①) and low-level specifications (Figure 3–②) of a bar chart (Figure 3–③) about *passenger_num* with *destination* in Table 1. Users can specify Figure 3–③ by Vega-Lite in Figure 3–①, and then Vega-Lite is compiled to Vega (Figure 3–②), finally users will get the target visualization (Figure 3–③).

Note that, in low-level languages, users have to specify the mapping function. For example, the "scales" in the Vega specification specifies the mapping function of the target visualization. The "xscale" denotes placing the categorical elements (*Atlanta*, *Boston*, *Chicago*, etc.) to the pixel range ($[0, 600]$, specified by "range":"width") of X-axis averagely. And the "yscale" denotes mapping the data range ($[0, 3500000]$, range of *passenger_num*) to the pixel range ($[0, 200]$, specified by "range":"height") of Y-axis linearly. But in high-level language, users only need to specify the mark type, e.g., bar, and do not need to specify the mapping function between data and mark. □

Note that, most of the low- and high- level languages listed in the survey are declarative languages (where the users only need to specify "what" they want) except Prefuse and Flare. Prefuse and Flare are procedural languages, because they are Java based visualization libraries, and users should initialize panels, add visual elements, etc.

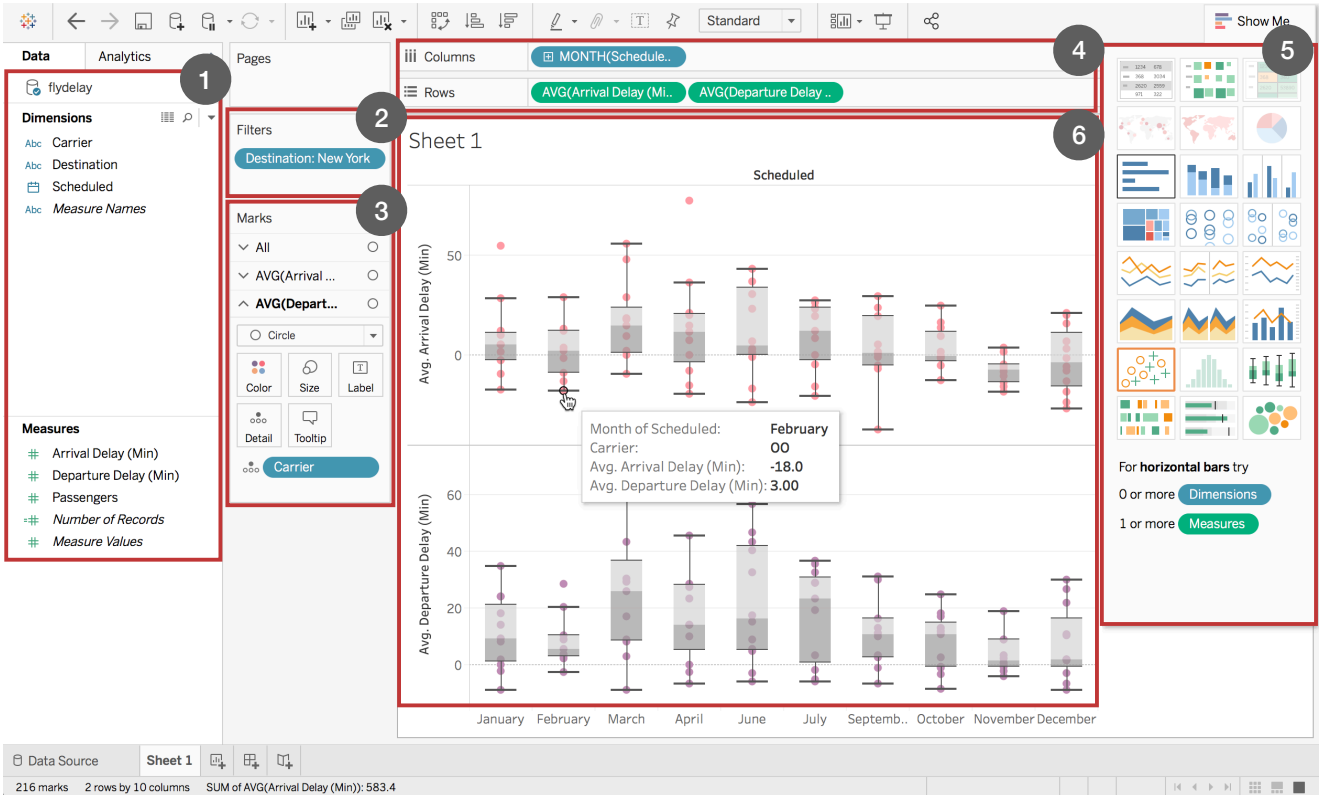


Fig. 4 An example of visual specification in Tableau using the flight delay data. ① displays the attributes of the loaded data, and users can drag attributes here to ④. ④ specifies the column attributes, row attributes, aggregation functions, and so on, for the specified visualization. The visualization of Tableau is in tabular structure. And the *Columns* and *Rows* in ④ denote the column attributes (i.e., *MONTH(scheduled)*) and row attributes (i.e., *AVG(departure delay (min))* and *AVG(departure delay (min))*) of the table. Users can choose the filter condition and visual mapping of marks in ② and ③ respectively. Also, users can click in ⑤ to specify the chart type. ⑥ displays the final specified visualizations to users, which is a box-and-whisker plot of average departure delay and average arrival delay with each month and carrier in 2017.

2.3 GUI-based Visual Operations

Compared with using declarative visualization languages to specify visualizations as discussed in Section 2.2, a more user-friendly way of providing a specification is to follow the "direct manipulation principle" [117], a widely used concept in the human-computer interaction aspect.

We have listed state-of-the-art GUI-based tools (Tableau [11], Qlik [10], Excel [7], Google Sheets [4], Amazon Quicksight [1], Microsoft Power BI [9], Google Fusion Tables [51], iVisDesigner [102], Lyra [108], Keshif [152], Data Illustrator [78, 79]) in Figure 2. Figure 4 shows an example of visual specification in Tableau using the flight delay data.

Remarks. Our main purpose of discussing GUI-based visual operations is to show different ways that users can specify visualizations. Regardless of using declarative languages or visual operations to specify visualizations, the common problems of making the process efficient and smart are the same. Hence, classifying the applications of different tools is out of the scope of this article, for which please see the slides[2] of Jeff Heer for an introduction of these tools.

Interactive Data Visualization. The rationality behind interactive data visualization is that in many cases, data visualization is a process of exploration, where the users need to keep refining the specification (e.g., add/remove/change attributes, change chart type) of current explored visualization until getting their desired visualizations in the exploration process.

We show two categories of interactive data visualization, Polaris and Tableau, using step-by-step query refinement to create multidimensional visualizations. Moreover, DeepEye and Voyager enable facet exploration and help users easily navigate the visualization.

(1) Stepwise Query Refinement. Polaris [125] and Tableau [11] provide chart templates to show multidimensional visualizations. Multidimensional visualiza-

[2] https://courses.cs.washington.edu/courses/cse442/17au/lectures/CSE442-Tools.pdf

Fig. 5 Faceted navigation in DeepEye: visualization ① is the root for exploration, and the suggested facets for visualization ① are *bin size* and *category*; Once the user chooses the facet *bin size*, she gets visualization ②. Visualization ② is different from ① only in the bin size (① and ② are binned by weekday and date respectively), and the other visualization elements (e.g., X-axis, Y-axis, chart type) of ① and ② are the same. Then DeepEye suggests 3 facets for ②: *chart type*, *category*, *bin size*.

tions are shown in two-dimensional plane organized in a tabular structure (Figure 4). Using tabular structure (e.g., 2 Rows × 1 Column in Figure 4) to display visualizations of different attributes (e.g., *AVG(departure delay (min))* and *AVG(departure delay (min))* in Figure 4) or different values of the same attribute is called "small multiples" [137], which is convenient to compare and analyze different attributes (different values of the same attribute). The "small multiples" are widely used in data visualization systems, such as Voyager [146], VizDeck [69], Show Me [84], Profiler [66], [45], [111], etc. Users can gradually drag multiple attributes to the rows, columns, layers of the tabular visualization, pick the appropriate visualization type, mapping of data to visual properties, etc., to build desired visualizations step by step.

(2) Faceted Navigation. DeepEye [99] supports faceted navigation to help users explore the visualization design space. Users can type in keywords, then DeepEye recommends relevant visualizations to users. Once a user chooses one interested visualization V, she can do a further navigation by different facets. The facets include *chart type*, *X-axis*, *Y-axis*, *category*, *bin size*, *group column*, and DeepEye will recommend visualizations which have the corresponding different facets with V while maintaining the other visualization elements unchangeable once users select one facet to explore. Also users can choose the *similar trend* or *different trend* facets, then DeepEye will recommend visualizations which have similar or different trend with V. Figure 5 shows a faceted navigation example on Table 1. Similar to DeepEye, Voyager [145] allows users to explore the visualization space by recommending visualizations which have the same or one more other attribute with current explored visualization.

Remark. Although GUI-based interactive tools provide simple interfaces to quickly construct common visualizations, which is of great importance for non-technical people, there may be limited chart types in the templates, and it is also not flexible to change details of visualizations, such as bar width, and color mapping, etc. Hence, in practice, similar to high-level visualization languages, GUI-based interactive tools are typically used for quickly prototyping, or for finding useful visualizations. Afterwards, low level languages (e.g., D3) will be used for fine tuning or reimplementing the desired visualizations.

2.4 Underspecified Specifications

Visualizations are meaningless if they cannot give insights of the data. However, in many cases, the users don't really know all aspects of the data at hand, because the data might be large and the data can be frequently updated. Hence, it poses a requirement of supporting underspecified specifications.

Generally speaking, for underspecified specifications, users only provide some "hint", and it is the task of the visualization systems to interpret the underspecified input, in (possibly) different ways.

The first type of hint is "reference-based", where the users provide a reference visualization as a seed and the system suggests visualizations based on the reference. zenvisage [118,119] returns similar or dissimilar visualizations (e.g., similar trends in line charts) with a user provided reference visualization.

The second type of hint is "keyword-based", in a Google style. APT [83] accepts user's data viewing goals of desired columns, for example, "*present the departure delay and scheduled relations*". In other words, APT specifies the *columns* to be visualized, and then recommends visualizations which satisfy the goals. DeepEye [82] is a recent system that accepts keyword inputs

as data viewing goals and provides recommended visualizations. For example, the user may input "*show me line charts about electricity*", and DeepEye will recommend line charts which also contain the column "electricity" to users. The demo of DeepEye can be found at `http://deepeye.tech`. A similar tool[3] that supports keyword inputs, called "Ask Data", was recently released by Tableau, which allows user to get answers without the need to know the structure of the data, such as "*what is the average price by variety*". SW [64] accepts users' window-based constraints (e.g., "*identify all windows in which the average departure delay* > 50") about desired visualization windows (a window is a rectangular region in a visualization).

The third type of hint is "natural language-based", which considers the context of user inputs and system states in the data exploration cycle instead of one-shot in "keyword-based" hint. Eviza [114] and Evizeon [59] are two recent visualization systems which provide natural language interfaces for visual analysis cycles. For example, in Evizeon [59], the user first types "show me the spike of measles in the UK", and Evizeon will show the user the spike in the line of measles outbreaks in the UK. Then the user types "mumps over there", and Evizeon will show the user the mumps outbreaks in the zone of the spike of measles.

Discussion. (1) We categorize visualization languages organized as a stack (see Figure 2), which is different from the survey [86] that categorizes visualization languages based on *graphic library*, *declarative*, *chart typology*, *data source*, *presentation medium*, and so on. (2) The survey [50] that evaluates different exploratory data analysis (EDA) tools for different applications is complementary to our survey, because we focus on how interactive data visualization tools construct visualizations through iterative interaction (i.e., *stepwise query refinement*, *faceted navigation*) with users.

3 Efficient Approaches for Data Visualization

In this section, we will discuss efficient approaches for data visualization; it is important because the data visualization life-cycle is always iterative (see Figure 1), with human-in-the-loop.

In the following, we will first describe *exact data visualization* that computes precise visualization as fast as possible (Section 3.1). Sometimes, however, providing exact visualizations may not always be doable because of the large size of data and high complexity of queries, *approximate data visualization* that provides fast but approximate visualizations is ideal for this case (Section 3.2). Furthermore, instead of only producing one-shot approximate visualizations, *progressive data visualization* gradually refines the intermediate results (Section 3.3).

Table 2 gives a summary of the techniques to be discussed in this section.

3.1 Exact Data Visualization

Many data visualization systems [81, 98, 132, 133] read data from databases. They may also manipulate data by SQL statements, and then use visualization tools to render the visualizations.

Query Translation. A natural way to reuse many mature (DBMS) systems is to translate the visualization queries to the queries those systems accept. For example, DeepEye [81,98], Polaris [125], SeeDB [132,133] get data by issuing SQL queries to the databases. By creating a mapping between the primitives of visualization language and SQL language, we can convert the target visualization language to a SQL query.

Example 2 The visualization f_1 in Table 3 specified by a ZQL query can be translated to a SQL query Q_1 as shown below.

```
Q1 : SELECT carrier, SUM(passengers)
     FROM flight delay
     GROUP BY carrier
     WHERE destination=‘‘New York’’;
```

The attributes of X - and Y - axes, i.e., *carrier* and *passengers*, can be mapped to the projection clause followed the keyword `SELECT`. The *Constraints* can be mapped to the filter condition following the `WHERE` clause. The *Viz*, i.e., *y = sum(passengers)* means that the SQL query should `GROUP BY` *carrier*, and apply *sum* to *passengers*. □

Integrating Visualization Systems with DBMSs. Although using query translation is natural, there are some disadvantages. One main issue is that many functionalities are repeated, resulting in non-unified optimization techniques with different assumptions and performance in server (i.e., the database side) and client (i.e., the visualization side), leaving the developers confused to choose the suitable optimization techniques. For example, the database engine and visualization tool may both support the *filter* operation; consequently, one can filter data records by either issuing a SQL query to database or by the function *filter* of JavaScript in the

[3] `https://www.tableau.com/about/blog/2018/11/ask-data-simplifying-analytics-natural-language-98655`.

Problem	Technique	Target
Exact Data Visualization	Query Translation [81, 98, 125, 132, 133]	Accelerate Visualization Exploration Process
	Integrating Visualization Systems with DBMS [147, 150]	
	Column Stores [66, 132, 133, 139]	
	Indexes [74, 95, 154]	
	Parallel Computation [77, 96, 132, 133]	
	Prediction and Prefetching [19, 31, 42, 64, 77]	
Approximate Data Visualization	AQP [40, 88]	Enable Quick Visualization Creation
	Incremental Sampling [48, 71, 101]	
	Human Perception [15, 71]	
Progressive Data Visualization	Hierarchical Aggregation [24, 44, 74, 77]	Enable Progressive Visualization Creation

Table 2 A summary of efficient data visualization, where we summarize the widely studied problems in efficient data visualization in column *Problem*, the corresponding techniques and references for each problem in column *Technique* and the target for solving the problem in column *Target*.

Name	X	Y	Constraints	Viz
f_1	carrier	passengers	destination= "New York"	bar. (y=sum(passengers))

Table 3 An ZQL query which returns a bar chart about the SUM(passengers) of different carriers to "New York", where *Name* denotes the visualization name specified by the ZQL query, X and Y denote attributes of the X- and Y-axes, *Constraints* specifies the constraints which the data used to generate the visualization should satisfied, and *Viz* specifies the visualization type and aggregation function on attribute of Y-axis.

front-end – choosing to *filter* at database or visualization tool (e.g., the front-end) is difficult. Another main issue is that decoupled methods are hard to maintain, extend and optimize [150] for interactive visualizations, which requires continuously issuing queries to modify visualizations.

Intuitively, a promising way to solve the above problems is to tightly couple (or integrate) data retrieval and rendering together to speedup the process of visualization creation. Ermac [147], a Data Visualization Management System (DVMS), is a research attempt on this direction. It supports two relations: `data` and `scales`, where relation `data` includes the data records to be visualized and references to the rendered visual elements; relation `scales` denotes the mapping from data ranges to visual encoding ranges. A visualization in Ermac is represented as a Logical Visualization Plan (LVP), and LVP is compiled into a SQL-like query. The SQL-like query deals with the `data` and `scales` relations, and the query constitutes a Physical Visualization Plan (PVP), then PVP can be optimized by the traditional database optimization techniques. During query execution, Ermac uses rendering placement and psychophysical approximation techniques to reduce latency. It also uses visualization features to support automatic lineage-based interaction, visualization estimation, recommendation, and so forth.

A further development [150] of Ermac is proposed to provide a SQL-like language, DeVIL, to represent both static and interactive visualizations. In DeVIL, *Marks* and *Pixels* are two visual relations to express visualizations which are expressed in SQL-like queries. DeVIL models the user inputs as event streams and database relations, and enables the interactive visualizations by executing SQL-like queries in joined visualizations and event relations iteratively to update the visualizations and response to user's inputs. By modeling the static and interactive visualizations as declarative database relations, visualization designers are released from event driven programming, making programming process more standardized and code more scalable. The work of [150] also proposes many optimization techniques (e.g., concurrency control and streaming framework) for interactive visualizations in DVMS.

Column Stores. In data management, a key performance factor is the data layout, e.g., row-based and column-based layouts, which may have a huge performance difference for OLAP applications. In terms of data visualization, the users are typically interested in only a few columns. Naturally, column-stores may achieve better performance, compared with row-stores, which have been adopted in SeeDB [132, 133], Profiler [66], and TDE [139].

Indexes. Indexes are widely used to improve search performance by essentially cutting down the number of records/rows in a table that need to be examined. Naturally, they play an important role in improving data visualization performance. FlashView [95] builds a hierarchical tree-based index to support users' selections with continuous filtering conditions. The work of [154] builds a tree-based index for the data which is to be queried instead of the whole dataset, and gradually refines the index when more data is queried. imMens [77] and Nanocubes [74] build datacubes which precompute aggregation results for different data slices

to reduce query execution time by accessing the precomputed aggregation results instead of the raw data. Hashedcubes [94] also builds datacubes for real time big data visualization. Hashedcubes uses pivot arrays to construct datacubes, while Nanocubes is tree-based. And Hashedcubes achieves lower memory usage and lower query time compared with Nanocubes. Gaussian Cubes [136] is a development of Nanocubes which supports more visualization analysis task types. For example, Gaussian Cubes precomputes sufficient statistics information in the datacubes to support model fitting.

Falcon [89] uses indexing techniques to reduce interaction time for brushing and linking in visualization. The visualization that the user is interacting (i.e., brushing) with is *active view*, and the other visualizations are *passive*. For current active visualization, Falcon builds index for each passive visualization. The index stores the data which should be highlighted in the passive visualization, and the data is in the form of array, where each entry of it stores cumulative counts. Thus Falcon can calculate the data to be highlighted in the passive visualizations in constant time given the start and end position in the active visualization. Since Falcon only maintains index for active visualization, it has much smaller index than imMens [77], Nanocubes [74], etc.

Parallel Computation. Parallel computation has also been widely used for query processing in data visualization systems [77, 96, 132, 133]. The aggregation queries on data tiles in imMens [77] are parallelized using the dense index representation of a data tile. SeeDB [132, 133] executes multiple SQL queries of visualization candidates in parallel during visualization ranking. Harald et al. [96] provide a multi-threading architecture for interactive visualization exploration. The architecture maintains a main application thread to capture users' interaction requests and multiple visualization threads for each visualization to process the visualization of this thread. Furthermore, whether the main thread and visualization threads are asynchronous or synchronous depends on the types of the users' interaction requests.

Prediction and Prefetching. One important step of data visualization is data exploration – users continuously browse their interested visualizations to get a sense of what to visualize. Oftentimes, the current explored visualization is usually inspired from the previous one. In other words, users may get the next visualization by changing parameters of current visualization or zooming in/out to get detailed/overall information, etc. Evidently, predicting the following data that users may be interested, and then prefetching/caching data which may be used in the next step during current exploration can speedup the exploration process, and these techniques have been used in many visualization systems [19, 24, 31, 42, 64, 77, 130].

We categorize the prefetch and prediction technologies to two types, based on:

1. Currently explored visualizations [24, 42, 64, 77], or
2. Historical data [19, 31, 41, 42, 153].

(1) *Currently Explored Visualizations.* XmdvTool [42] clusters tuples in different granularity to support users' hierarchical navigations. It enables users to continuously explore data in the structured-based brush [49]. Hence, it needs to predict the next direction of the user, and then prefetches and caches the data in that direction during the idle time. The caching system uses the least recently used (LRU) as the replacement policy and the current explored visualization based prefetching strategy is to randomly pick a direction from the position of the current explored data.

Following the above hierarchical navigations, instead of prefetching only one piece of data on the tree hierarchies, it is also natural to prefetch different levels' representation of the present data, as used in imMens [77] and [24].

Another angle for a good prefetching is based on the size of prefetched data (e.g., SW [64]), instead of the direction on the hierarchy that the user will explore. More specifically, SW finds all windows that satisfy users' constraints inputs (e.g., "*identify all windows in which the average departure delay* > 50"). SW iteratively explores all possible visualization windows to find "good" (i.e., satisfy users' constraints inputs) windows. When a window is being explored, SW prefetches the neighbor windows in all directions, but the size of the data to be prefetched in each direction should be decided by the algorithm. SW first computes whether the prefetched window satisfies the constraints by sampling data of the window. If the result is true, the prefetching in this direction is stopped, and the prefetched window is to be explored further (the exploring process is same as the current window). Otherwise, SW continues to prefetch in that direction by increasing the sampling rate until the data in this direction are all prefetched or the constraints are satisfied. Note that SW wants to find all "good" windows in the search space, thus it must explore all possible windows, and by exponentially increasing the sampling size, SW can terminate the exploration quickly with less prefetching times, thus getting all "good" windows with less time.

(2) Historical Data. Next, we will discuss techniques that leverage historical trajectories [19, 31, 42, 153] for prefetching.

When historical data is available, naturally, systems can do more complicated yet meaningful inference than randomly picking a direction as discussed above. More specifically, XmdvTool [42] proposes three strategies to prefetch the data based on historical data:

- the *direction*: select the most likely direction based on the users' previous trajectory tracking,
- the *focus*: select the direction with hot regions, and
- the *vector*: select the direction based on the vectors of the movement trajectories of the users, in the form of <*start position*, *width*, *level*>, where *start position* is the start location and orientation of the movement, *width* is the moving distance of the movement, *level* is the aggregation hierarchy of the data explored in the movement. It uses the mean or exponential weighted average of previous trajectory vectors to select the directions.

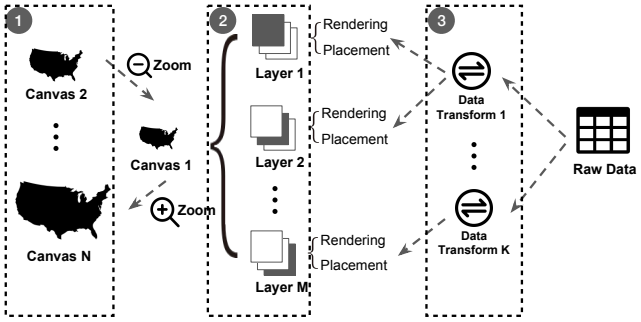
Recently, machine learning based approaches have also been studied. ForeCache [19] partitions data to blocks or data tiles in different levels, and predicts data tiles to users. There are two stages of data prediction:

- *Predicting analysis phases:* it predicts the users' exploration phase by a Support Vector Machine (SVM) model, and the features include position, panning and zooming information of users' exploration traces.
- *Predicting data tiles:* it uses the corresponding strategies to recommend prefetched data: ① action based strategy using Markov chain which accepts sequences of users' movement (e.g., {*left, left, left*}) as a state and a move from state to state as a transition (e.g., "*right*"); ② signature based strategy which recommends similar data tiles with users' previous explored data.

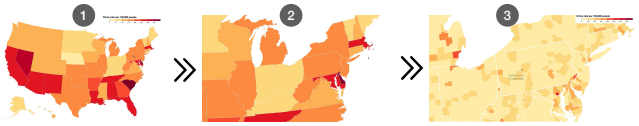
Experiments have shown that ForeCache achieves 25% higher prediction accuracy than the prediction strategies in XmdvTool [42].

Case Studies using Kyrix and Tableau. In the following, we will discuss two case studies using Kyrix, an interactive scalable data visualization system, and Tableau, one of the most successful visualization tools.

Kyrix [129] is an interactive scalable data visualization system. Kyrix provides declarative visualization specification interface in front-end and effective scalable visualization processing in back-end, where the user zooms in to see detailed information and zooms out to overview in scalable visualization.



(a) Declarative model of Kyrix [129]. A canvas (①) contains multiple layers (②), and a layer can be got by specifying a rendering and placement function for the transformed data (③).



(b) Zoomable crime rate map of US by Kyrix [129]. ① is the original visualization, and ②,③ are more detailed visualizations by zooming in.

Fig. 6 Declarative model and zooming example of Kyrix [129].

1. *Visualization Specifications in Front-end.* There are two abstractions in the visualization specification language of Kyrix: *canvas* and *jump*. A *canvas* contains a static visualization, where the data of the visualization is specified by a SQL query and the transformation and rendering function can be specified by existing visualization libraries (e.g., D3 [87], Vega [12]). A *jump* specifies the source and destination canvas and the transition type when panning or zooming.
2. *Efficient Approaches for Data Visualization in Back-end.* There are two important improvements in Kyrix: *fetching granularity* and *indexing*. For *fetching granularity*, Kyrix splits raw data to static data tiles of fixed size. The tiles of current visualization together with a dynamic box which encompasses these tiles are sent to the front-end, and the box is recalculated when the tiles of current visualization are out of it. Compared to fetching large or small tiles, Kyrix can adjust the size of dynamic box by different algorithms, providing a way to neutralize the network time and prefetching size. For *indexing*, Kyrix builds Btree [20] or hash indexes on the tile id of a tuple to support quick fetching.

Figure 6(a) shows the declarative model of Kyrix. A *canvas* (Figure 6(a)–①) in Kyrix is a level of detail of data, and users can zoom in/out to see more canvas of different levels of details. A *canvas* may contains more than one *layer* (Figure 6(a)–②) (e.g., background layer, line layer, etc.), and there should be a *rendering*

function and *placement function* for the transformed data (Figure 6(a)–③) of each *layer*, where the *rendering function* defines how to map data to visual objects, and the *placement function* gives the location of the visualized data by the *fetching granularity* and *indexing* strategies. Figure 6(b) is a zoomable crime rate map of US by Kyrix.

TDE [139] is a data engine customized for visualization in Tableau 6.0. TDE optimizes the data engine mainly in the following perspectives.

1. *Column-oriented Storage and Compression.* Due to the high I/O cost of Tableau's former database Firebird and data of visualizations usually stored in different columns, column-oriented storage and compression techniques have been designed to solve this problem in TDE. TDE mainly uses dictionary compression strategy, and there are two compression mechanisms for dictionary compression: *heap compression* for variable width types and *array compression* for fixed width types. Then the compressed columns can be represented by the *dictionary tokens* (i.e., the dictionary keys) which reference the dictionary values during the query execution.
2. *Operator Reordering.* Selection operators and operators with single compressed columns are pushed down in the SQL query plan tree.
3. *Cardinality Reduction.* For columns with high cardinality columns, TDE automatically transforms these columns to higher hierarchies, e.g., transforms column *Time* with 2500 distinct values to column *Year* with 7 distinct values, then pushes the operators with *Time* in SQL query plan tree down and replaces *Time* with *Year*.
4. *Other Visualization Support.* TDE provides domain information (e.g., the cardinality, maximum and minimum values of the domain) of columns. This domain information can be used to choose the level of detail of a visualization for users. TDE also supports progressive reporting and termination control (i.e., terminate long running visualization queries) when executing visualization queries.

The recent effort of Tableau 10's server data engine is to customize a highly efficient main-memory system Hyper [5, 92, 93]. Hyper is used as the data engine to power all versions of Tableau, such as Tableau Server, Tableau Desktop, Tableau Online, and Tableau Public. In particular, Hyper is used to support efficient creation, refresh, query extraction, and cross-database joins.

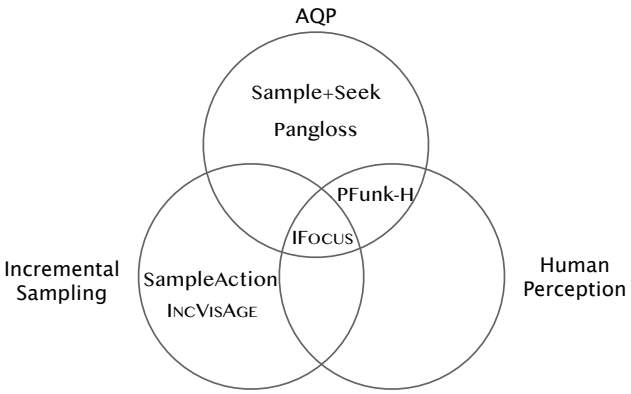


Fig. 7 A classification of approximate data visualization, where the surveyed works are classified as AQP, Incremental Sampling, and Human Perception methods.

3.2 Approximate Data Visualization

When the data volumes grow exponentially, traditional data processing modules cannot provide fast interactive processing results. To bridge the gap between data volumes and interactivity, many works [15, 40, 48, 71, 88, 101] speedup data processing phase by leveraging Approximate Query Processing (AQP) that provides approximate visualization results.

We discuss approximate data visualization from three perspectives: *AQP*-based approaches that leverage techniques from AQP; *incremental sampling*-based approaches that link incremental query processing to visualization; and *human perception*-based approaches that capture the cognitive limitations of human perception. A summary of the contents to be discussed is shown in Table 4 and Figure 7.

AQP-based. A straightforward way for generating approximate visualizations in interactive time is leveraging the techniques of AQP. Using the representative subset of the data can provide users with approximate visualizations for online interaction by sacrificing the quality. We will review two works [40, 88] that mainly focus on the sampling-based AQP techniques.

Sample+Seek [40] is an AQP system for answering visualizations generated from aggregation queries in an interactive speed, and the visualization results are within an error bound specified by users. It first presents the concept of *distribution precision* (e.g., distance between the approximate and exact visualizations) that can represent the precision of total distribution across aggregate groups. Thus, users can specify a *distribution precision* as an error bound. When sampling, for those queries with large data volumes, it uses the *uniform sampling* to answer the `COUNT` aggregation queries and proposes a *measure-biased sampling*

Paper	Algorithm	Visualization Types				Query Types					
		Bar	Pie	Line	Heatmap	GROUP BY	WHERE	ORDER BY	COUNT	SUM	AVG
Sample+Seek [40]	Uniform Sampling & Measure-biased Sampling	√	√	×	×	√	√	×	√	√	√
Pangloss [88]	Optimistic Visualization Based on AQP	√	×	×	√	×	×	×	×	×	×
SampleAction [48]	Sampling-based Incremental Visualization	√	×	√	×	√	√	√	√	√	√
IncVisAge [101]	Sampling-based Incremental Visualization	×	×	√	√	√	√	√	√	√	√
IFocus [71]	IFocus Algorithm	√	×	√	√	√	√	×	√	√	√
PFunk-H [15]	Human Perceptual Model with Sampling-based AQP	√	×	×	×	×	×	×	√	√	√

Table 4 Summary of approximate data visualization systems, where we summarize the papers and algorithms for approximate data visualization systems in column *Paper* and *Algorithm* respectively, and the supported visualization and query types of each algorithm in column *Visualization Types* and *Query Types*.

technique for approximately answering `SUM` aggregation queries with less predicates, and the key feature of *measure-biased sampling* is to select the rows with probability proportional to its value on the aggregation attribute. Sample+Seek proposes two indexing techniques to speedup sampling: *measure-augmented inverted index* for indexing the categorical dimension to answer the aggregation queries; and *low-frequency group index* for supporting those queries with a conjunction of one or more equi-constraints.

Although there exists a significant difference between approximate and accurate visualizations with a small possibility, users may get frustrated with the visualization tools once a big difference happens. Thus Pangloss [88], a web-based system powered by Sample+Seek [40], is designed to provide users with approximate visualizations together with exact visualizations. Pangloss provides users with approximate visualizations quickly based on the technique of AQP and then the system still computes the exact results in the background if users click the "remember" button for this visualization. In Pangloss, users can get initial insights from the approximate results and later verify their observations on the precise results.

Incremental Sampling-based. Some works [48, 71, 101] try to link incremental data query techniques to data visualization. The key idea of approximate visualization with incremental sampling is that the system generates an approximate visualization based on representative samples of dataset rapidly. Then, the system increases the sample size over time to continuously improve the quality of visualizations. The user usually can get some initial insights from the approximate visualizations and decide to terminate if the quality of the visualization is enough to verify these insights.

SampleAction [48] is a tool for visualizing aggregation queries on very large datasets. Given a query, SampleAction rapidly responds to users with partial aggregation results for each group with error bounds (i.e., a bar chart with confidence bounds) based on fixed samples. As the users are waiting, it will narrow its error bound and incrementally improve the visualizations by increasing sampling size in every second.

There may exist significant fluctuations between the adjacent incremental approximate visualizations due to the random sampling in SampleAction [48], which may mislead users during the incremental approximation process. Thus IncVisAge [101] is designed to solve this problem. IncVisAge [101] is a web-based system, which provides incremental approximate visualizations, typically supporting trendline and heatmap. And there are no significant fluctuations compared with SampleAction [48] during the process of visualization refinement due to the design of the *ISplit* algorithm, thus providing meaningful intermediate visualizations for users. In IncVisAge, a trendline displays aggregation results for all groups, and a segment denotes multiple successive groups of the trendline together with the same approximate aggregation value. A trendline is first initialized as a segment with all groups, and during the iterative process, the *ISplit* algorithm chooses one segment, and splits the segment into two segments until there is no segment to split (i.e., all segments have only one group). During each iteration, the *ISplit* algorithm calculates the sampling number for given δ (failure probability) and ϵ (guarantee of error), and then chooses the best segment to be split based on the sampled data.

Figure 8 illustrates what approximate visualizations might be generated by SampleAction and IncVisAge, where significant fluctuation exists during two adjacent visualizations generated by SampleAction while IncVisAge keeps stable updating. For example, in Figure 8–①, the visualizations generated by SampleAction at t_1 and t_2 have very different trends. But

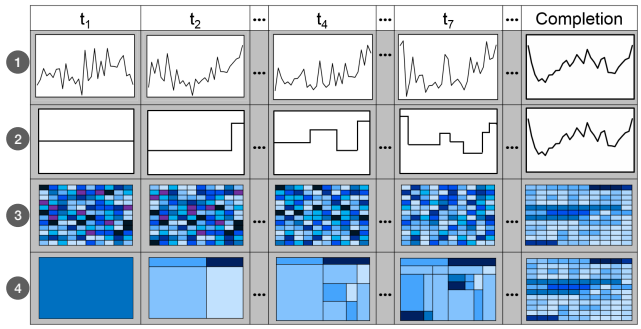


Fig. 8 A schematic diagram (not generated by real data) showing what approximate visualizations might generated by SampleAction and IncVisAge as time progresses (i.e., more samples are sampled) [101]. ① and ② are approximate lines generated by SampleAction and IncVisAge respectively using the same data, and ③ and ④ are approximate heatmaps generated by SampleAction and IncVisAge respectively using the same data.

IncVisAge only splits one segment to update visualizations, thus keeps stable updating.

Human Perception-based. At times, increasing the sample size does not always improve the quality of the visualization. The external reason is that the number of pixels of the screen is finite and the internal reason is that the cognitive limitations of human perception in identifying small details. Therefore, it is possible for approximate visualization systems to generate approximate results based on representative samples but with minimal impact on the quality of visualization. Human perception-based approaches stop sampling when there is no obvious difference on human perception between the current approximate visualization and the visualization which is to be get by further sampling.

IFocus [71], an online sampling algorithm, can generate an approximate bar chart rapidly and guarantee the pairwise ordering of each bar in a bar chart, because the pairwise ordering in bar charts are important human perceptual focuses. IFocus iteratively draws a sample for each *active* group (all groups are *active* at first), and maintains a confidence interval for each *active* group. Once a confidence interval of a bar has no overlap with other bars, meaning the order of this bar is determined, i.e., this bar is not *active* any more. The algorithm terminates until all bars are not *active*, and the approximate visualization results with ordering guarantee are returned to users.

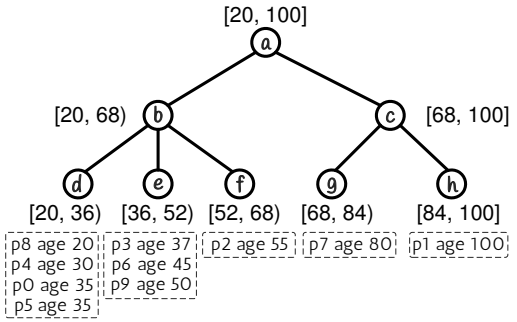
PFunk-H [15] addresses the approximate visualization as the human perception problem. The basic idea of this work is that to combine sampling-based AQP techniques and human perception limitation together to provide approximate visualization in order to satisfy human perception. PFunk-H is an online sampling algorithm that provides approximate visualizations using perceptual functions from graphical perception. It presents an algorithm that can learn the knowledge of human perception error to provide approximate visualizations with perceptually indiscernible error. Besides, it can provide error bound of approximated query results under the restriction of perceptual function.

3.3 Progressive Data Visualization

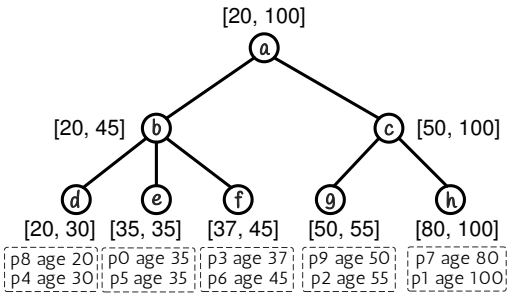
Many works [48, 71, 101] in approximate data visualization (Section 3.2) produce progressive visualization results to users. Besides the above incremental sampling based progressive data visualization, there have also been many works [24, 44, 74, 77] which provide progressive visualizations by hierarchical aggregation. Generally speaking, they build a hierarchical structure by aggregating the data in different levels, for example, different sizes of bins, different ranges of temporal values, different zones of spatial values. Then these hierarchical structures are used to support users' progressive visualization exploration.

Range Based Binning. imMens [77] provides visualizations of different resolutions by changing bin sizes. The bins of the same resolution have equal ranges. Multi-dimensional data in imMens is partitioned into data cubes, and cubes are partitioned into tiles of different levels. Users can explore data in different levels and change current explored visualizations' resolution by zooming in or zooming out, then the system will change the underlying aggregation bin size correspondingly. imMens bins numeric data by equal ranges, which has some limitations. For example, considering an examination transcript dataset, binning by equal ranges of score is not applicable if teachers want to know the top-10, top 10 to 20, top 20 to 30 students, etc. Also, users cannot change the bin size or number of different resolution in imMens.

Range and Content Based Binning. The work of [24] provides two tree-structures for hierarchical exploration: HETree-R (Range-based HETree) and HETree-C (Content-based HETree). HETree-R is similar with imMens, and the leaf nodes of HETree-R denote data points within equal width ranges, while HETree-C has the same number of data points in all leaf nodes. Thus the HETree-C can be used in the above examination transcript scenario. Users can explore the data abstraction or data details by a roll-up or drill-down operation to reach the upper or next level. It provides incremental tree construction algorithms based on user interaction and exploration scenarios (top-down or bottom-up). The algorithms automatically determine the proper arguments of the tree, i.e., the height of the



(a) HETree-R.



(b) HETree-C.

Fig. 9 A binning example of HETree-R and HETree-C. Ten points (p0 – p9) are binned by attribute age. HETree-R bins data by equal age ranges: each leaf is a bin with size 16, and HETree-C bins data by equal number of points: each leaf is a bin with 2 points.

tree, the range of the leaf, the number of children, etc. Also, the tree construction algorithms are adaptive and can fit to users' preference selection for parameters of tree to support better user experience. Figure 9 is a binning example of HETree-R and HETree-C.

4 Visualization Recommendation

Recall that in Figure 1, the data visualization process is iterative, and the main pain point of practitioners is that they have to be involved in each step to make some modifications. Naturally, it is highly desirable that there can have some visualization recommendation solutions that make the lives of users easier, by recommending (possibly) good visualizations to them.

The rest of this section will be organized as follows.

- Specification-based recommendations. (Section 4.1)
  - The specification is incomplete, i.e., empty or partial specification of visualization elements. (Section 4.1.1)
  - The specification is treated as a reference. (Section 4.1.2)
- Behavior-based recommendations. (Section 4.2)
- Personalized recommendations. (Section 4.3)

Solution Overview. Generally speaking, for solving all the above problems, the visualization recommendation systems need to first enumerate all possible visualizations and then recommend top-ranked visualizations.

Note that the search space of all visualizations is huge, which needs to consider the combination of several factors, such as choosing the columns to be visualized, transforming the data (e.g., group or bin), choosing the right visual encodings including mark types (e.g., bar, line, point), and encoding types for the selected mark chart (e.g., width of bar, position of point).

Pruning Meaningless Visualizations. Fortunately, there are many signals (or constraints) – either from the users or from traditional wisdom – that can be used to prune "bad" visualizations.

- *User specified constraints.* Users can specify interested visualization elements such as columns or data records.
  - SeeDB [132] stipulates that user should specify an interested query Q to obtain the target data before recommendation.
  - In Voyager [144–146], users should first specify interested variables too.
- *Expert provided constraints.* Some combinations of variables, transformations and visual encodings may not generate a valid visualization. For example, the mark type "pie" cannot combine with the encoding type "height", and the encoding type "Y-axis" is not suitable for categorical attributes. These constraints are typically given by experts.
  - Voyager [144–146] develops a permitted combination table of different data types, encoding types and mark types.
  - DeepEye [81,82,98,99] defines a set of rules (Table 5) to generate meaningful visualizations. For example, the first transformation rule in Table 5 denotes that if X-axis of a visualization is categorical, and Y-axis is numerical, then the transformation operation should be grouping by X-axis, and aggregating on Y-axis.
  - Draco [90] has hard constraints and soft constraints, where hard constraints must be satisfied when generating visualizations (e.g., the encoding type "shape" is not applicable for numeric values), and soft constraints are used to rank visualizations (e.g., it is better to use the encoding type X-axis for temporal values).

After generating candidate (or valid) visualizations by pruning the entire search space as described above, visualization recommendation systems will then recognize meaningful visualizations based on predefined met-

Transformation Rules.
***T**(X) = Categorical, **T**(Y) = Numerical → **GROUP**(X), ***AGG**(Y).
T(X) = Categorical, **T**(Y) ≠ Numerical → **GROUP**(X), **CNT**(Y).
T(X) = Numerical, **T**(Y) = Numerical → **BIN**(X), **AGG**(Y).
T(X) = Numerical, **T**(Y) ≠ Numerical → **BIN**(X), **CNT**(Y).
T(X) = Temporal, **T**(Y) = Numerical → **GROUP/BIN**(X), **AGG**(Y).
T(X) = Temporal, **T**(Y) ≠ Numerical → **GROUP/BIN**(X), **CNT**(Y).
Sorting Rules.
T(X) = Numerical/Temporal → **ORDER BY**(X).
T(Y) = Numerical → **ORDER BY**(Y).
Visualization Rules.
T(X) = Categorical, **T**(Y) = Numerical → **BAR/PIE**.
T(X) = Numerical, **T**(Y) = Numerical → **LINE/BAR**.
T(X) = Numerical, **T**(Y) = Numerical, (X,Y) correlated → **SCATTER**.
T(X) = Temporal, **T**(Y) = Numerical → **LINE**.
***T** = Type, **AGG** = {**AVG**, **SUM**, **CNT**}

Table 5 Constraints in DeepEye (**T** denotes the data type, **AGG** denotes the aggregation function, including **AVG** (average), **SUM** (sum), **CNT** (count), and X, Y denote attributes for X- and Y- axes respectively). The transformation rules define how to transform the data when the data types of the X- and Y- axes are given. The sorting rules define how to sort the data. And the visualization rules define how to choose the right visualization types for different data types of X- and Y- axes.

rics or rules. Some systems may also rank interesting visualizations or recommend top-k visualizations to users. In the rest of this section, we will discuss these methods to solve the above problems.

4.1 Specification-based Recommendations

4.1.1 Incomplete Specification

Visualization recommendation systems with *empty specification* require no user inputs, while recommendation systems with *partial specification* accept users' partial visualization elements specification inputs for desired visualization. For example, APT [83] accepts user's data viewing goals before recommendation. Users should first choose one interested column before visualization recommendation in Voyager [144–146]. DeepEye [81, 82, 98, 99] accepts users' keyword specification, e.g., the user may input "*show me line charts about electricity*".

The only difference between empty and partial specification is that the latter should prune the search space by the user specified constraints when enumerating visualization elements to generate visualization candidates. For example, the visualizations which do not contain column c are filtered from the visualization candidates if users specify column c as the interested column in Voyager, and the visualizations which are not line charts or do not contain the column "*electricity*" are filtered from the visualization candidates when users type "*show me line charts about electricity*" in DeepEye. And there is no difference when ranking candidate visualizations.

In the remaining part of this section, we describe two common methods used to rank the visualization candidates: rule-based solution and machine learning-based solution.

Rule-based Visualization Ranking

Most earlier works (APT [83], SAGE [105], BOZ [30]) on visualization recommendations are rule-based, which are inspired by the work of [22, 34, 83, 116, 124]. Rule-based recommendation systems rank the visualization candidates by their predefined rules, which are usually human perceptual effectiveness metrics, measured as an effectiveness score s considering data type, statistical information, human visual preference, etc. For example, a pie chart consists of many blocks (e.g., > 500) is not a good visualization by human perception, because it is too messy. The main difference in the rule-based recommendation systems is the definition of s. Voyager [144–146], Show Me [84], Polaris [125], DIVE [61], DeepEye [81, 82, 98, 99], Wang et al. [135] develop richer perceptual rules with more data types, mark types, statistical information compared with former works, while Rank-by-feature [113] ranks visualizations by a single statistical metric.

Statistical Rules. Rank-by-feature framework [113] is a statistical rule-based recommendation system. It can rank 1D or 2D axis-parallel projection visualizations (histograms, boxplots and scatterplots) to users by different statistical ranking metrics. The metrics for 1D ranking (histograms and boxplots) include normality or uniformity of the distribution, number of potential outliers or unique values and size of the biggest gap, and the metrics for 2D ranking (scatterplots) include correlation coefficient, number of potential outliers, uniformity of scatterplots, etc. By discovering these ranked low-dimensional visualizations, users may find complex relations, clusters, outliers and so on.

Perceptual Rules. The Rank-by-feature framework can only rank between the same visualization type (e.g., histograms, boxplots) by a single statistical metric, while Voyager [144–146] ranks different visualization types by a perceptual effectiveness score s considering data type, cardinality, human visual preference and so on. For example, high cardinality variables should not be mapped to color; visualizations with less screen space are preferred. s is a weighted sum of these factors, and the weight of these factors are manually determined through tests and experiments. Table 6 shows some perceptual effectiveness ranking rules used in Voyager, for

T(X)	T(Y)	Mark Type
Categorical	Categorical	point > text
Categorical	Numerical	bar > point > text
Temporal	Numerical	line > bar > point > text
Numerical	Numerical	point > text

Table 6 Ranking rules in Voyager. *T(X)* and *T(Y)* denote the data type of X- and Y- axes respectively, and *Mark Type* denotes the permitted ranked mark types for this data type correspondingly.

example, the third row in Table 6 means that if the data types of X- and Y- axes are temporal and numerical respectively, then line chart is the best choice, and bar, point, text types ranked behind line. And the rules together with users' input for column preference form the ranking metric in Voyager.

DeepEye [81, 82, 98, 99] captures human perceptual effectiveness in richer details than Voyager. DeepEye defines three factors to describe the quality of a visualization, then develops a partial order based solution to rank all the valid visualization candidates. The three factors are: ① the matching quality between data and chart; ② The quality of transformations; and ③ the importance of columns. A visualization precedes another if all of the three factors is greater than another. And based on the partial relation, DeepEye can construct a graph $G(V, E)$, where V denotes the all valid visualizations and E denotes the partial orders. Then DeeyEye ranks all the visualizations in a way similar to topological sorting.

The above two works (Voyager [144–146] and DeepEye [81, 82, 98, 99]) consider common visualization types, while Wang et al. [135] propose an algorithm to automatically pick line graph or scatter plot for time series. Although people may use line graphs to visualize time series in most cases, scatter plots are better choices sometimes. For example, the scatter chart provides a clearer trend than line chart when there are many outliers in time series. The algorithm first constructs line graph, scatter plot and a trend curve by LOESS regression [33], then calculates the visual consistency between the trend curve and the line graph or scatter plot respectively, and picks the visualization type which has bigger visual consistency (i.e., smaller distance) with the trend curve, where the visual consistency is achieved by comparing the consistency (i.e., distance) of visualizations' density fields, which can be calculated by KDE [120] algorithm. Experiments have shown that the choices of the algorithm are consistent with users in most cases.

Machine Learning-based Visualization Ranking

With the rapid development of machine learning and deep learning, more and more systems [39, 69, 82, 90, 98] focus on machine learning-based visualization recommendation. Given two visualizations u and v, the systems should determine which is better. Typically speaking, machine learning-based recommendation systems first collect training data, which comes from crowdsourcing or web, then train a ranking model which takes the input space $\mathcal{X}$ as lists of feature vectors, and $\mathcal{Y}$ the output space consisting of grades (or ranks). The model learns a function $F(\cdot)$ from the training examples, such that given two input vectors $\mathbf{x_1}$ and $\mathbf{x_2}$, it can determine which one is better, $F(\mathbf{x_1})$ or $F(\mathbf{x_2})$.

Learning with Soft Constraints. Draco [90] expresses preferences by soft constraints, and the soft constraints (e.g., it is better for temporal values to use the encoding type: X-axis) are specified by human perceptions. Each soft constraint has a weight denoting the penalty when a visualization v violates the soft constraint. And the overall cost of V is: $cost(v) = \sum_{i=1}^{n} w_i \cdot n_i$, $n_i \in \{0, 1\}$, where w_i is the weight of the ith constraint, and n_i denotes that V violates the ith constraint n_i times. Draco prefers visualizations with less cost, and formulates the problem of learning weights as a *learning-to-rank* [28] problem using RankSVM model [56]. Draco gets 1110 ranked visualization pairs from crowdsourcing, and the ranking principles are from Kim et al. [72] and Saket et al. [106]. Draco is similar to Voyager, but Draco differs with Voyager in that it learns the weight of constraints by machine learning techniques.

Learning with Examples. The constraints in Draco are pre-defined to the system by users or developers, rather than learned by machines. In contrast, DeepEye [82, 98] develops a machine learning-based solution which captures visualization design knowledge automatically by learning from examples besides the above rule-based solution.

DeepEye [82, 98] captures human perception by learning from examples, and supposes the models learned from former examples can be extended to different domains. DeepEye identifies 12 features: statistical information (cardinality, distinct numbers, max, min, correlation, etc.) and mark type. DeepEye uses a binary classifier (decision tree [100]) to determine whether a visualization is good or bad, which is called *Visualization Recognition*, then use a *learning-to-rank* [28] model (LambdaMART algorithm [151]) to score all good visualizations, which is called *Visualization Ranking*. DeepEye collects 42 real-world datasets from various do-

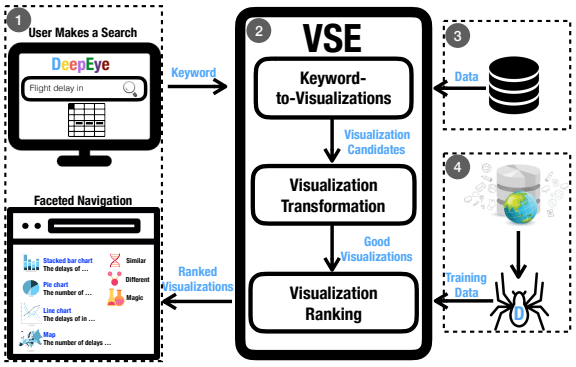


Fig. 10 The architecture of DeepEye [82,98]. User can post a keyword search in ①, and ② generates visualization candidates by querying ③, then ② returns ranked visualizations to user using the model trained by data from ④.

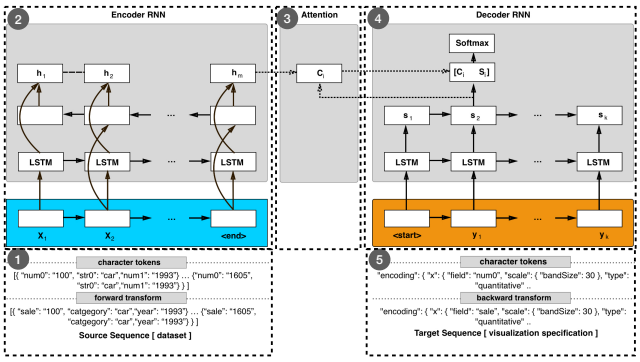


Fig. 11 The architecture of Data2Vis [39]. Data2Vis is built based on sequence to sequence model with the encoder-decoder architecture (② and ④) and the attention mechanism (③). It takes original datasets (①) as input, and automatically recommends visualizations (⑤) by given datasets (①).

mains, then picks 285,236 visualization comparisons over these datasets labeled by 100 students.

Figure 10 shows the architecture of DeepEye. A user can pose a keyword query to VSE (Visualization Search Engine) module (Figure 10–②), and VSE returns ranked visualizations to users. The VSE first translates the keyword query to multiple visualization candidates (keyword-to-visualizations) by querying database (Figure 10–③), then discovers good visualizations (visualization transformation) and ranks them (visualization ranking). The crawler (Figure 10–④) extracts training data for visualization transformation and ranking of VSE. In the client (Figure 10–①), users can input keyword, interact with recommended visualizations, and do faceted navigation, etc.

Data2Vis [39] is an attempt in generating visualizations using RNN (recurrent neural network [46]). As shown in Figure 11, Data2Vis treats visualization design as a sequence to sequence [16, 127] translation problem, where the input string is a dataset in JSON format (Figure 11–①) and the output string is a Vega-Lite [109] visualization specification (Figure 11–⑤). Data2Vis trained a model with a 2-layer RNN encoder (Figure 11–②) and a 2-layer RNN decoder (Figure 11–④), and both have 256 LSTM (Long Short-Term Memory [58,126]) cells. The training dataset has 4300 training instances [97]. Experiments have shown that Data2Vis can generate visualizations with appropriate mark types (e.g., use scatter for two numeric attributes), transformations (e.g., use means for numeric attributes), selection patterns (e.g., select data by country, gender), etc.

4.1.2 Reference-based Specification

Some visualization recommendation systems recommend visualizations based on reference data or reference visualizations [66, 118, 119, 132]. Typically, the system would recommend visualizations which are similar to or different from the given reference in certain aspects.

Deviation-based. SeeDB [132] recommends visualizations by deviation with some reference visualizations. Before recommendation, the user should specify an interested query Q to obtain the target data which is called D_Q and a reference dataset D_R is also needed. Then SeeDB enumerates different combinations of same variables, transformation, mark types, encoding types on both D_Q and D_R to get all $V(D_Q)$ and $V(D_R)$ respectively. Finally SeeDB recommends the top-k $V(D_Q)$ which have the largest value $S(P[V(D_Q)], P[V(D_R)])$, where S is a distance function, and $P[V(D_Q)]$ and $P[V(D_R)]$ are the distribution of $V(D_Q)$ and $V(D_R)$ respectively. Figure 13 shows the architecture of SeeDB.

Anomaly-based. Profiler [66] recommends visualizations which can best distinguish anomalies in the primary visualization. The tuples in the primary visualization are classified by some anomaly detection methods: normal points and abnormal points are in different classes, denoted as a column *class*. Suppose $VisToCol(V)$ is a function which returns a column that describes the classes of each tuple in the visualization V, then Profiler recommends the visualization V that minimizes $D(VisToCol(V), class)$. $D(X, Y)$ is a distance function measuring the independence between X and Y:

$$D(X,Y) = 1 - \left(\frac{I(X,Y)}{\max\left(H(X), H(Y)\right)}\right) \quad (1)$$

where $I(X,Y)$ denotes mutual information of X and Y, quantifying the reduction of predicting one variable when another is given, and $H(X)$, $H(Y)$ denote entropies of X, Y respectively.

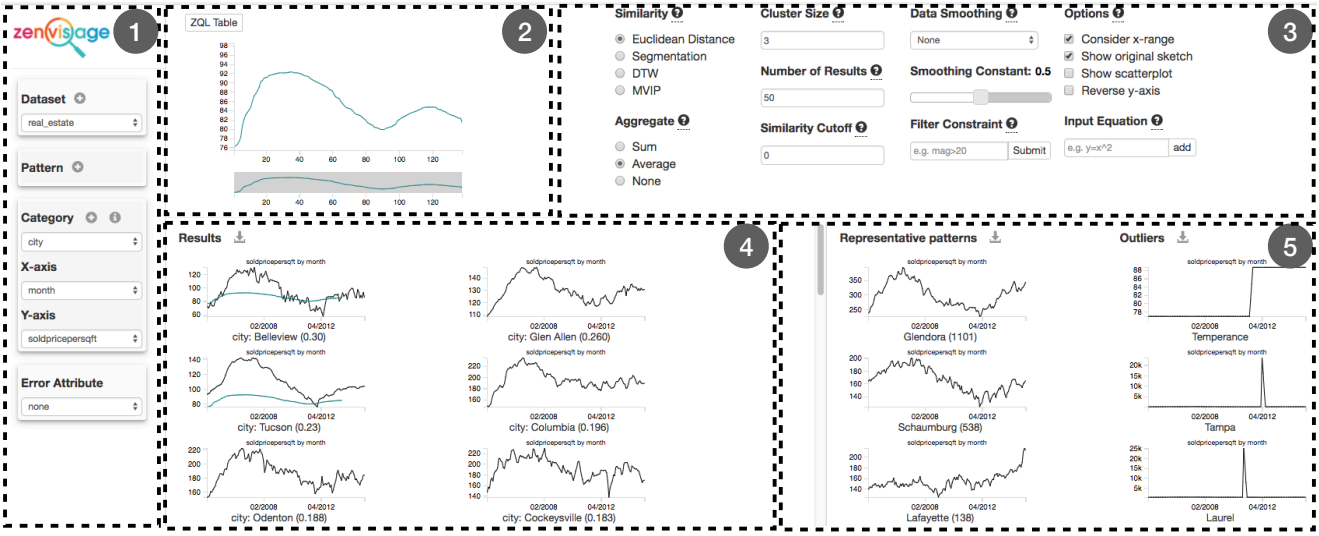


Fig. 12 The frontend of Zenvisage [118, 119]. The user can upload their datasets, and select the X-axis, Y-axis, and category for visualizations in ①. After that, Zenvisage first recommends representative (i.e., typical trends) and outlier trends in ⑤ according to the settings of ①. ④ will also show those visualizations that are similar to the reference one in ②. The user can specify some system parameters, e.g., similarity functions and aggregation functions, in ③.

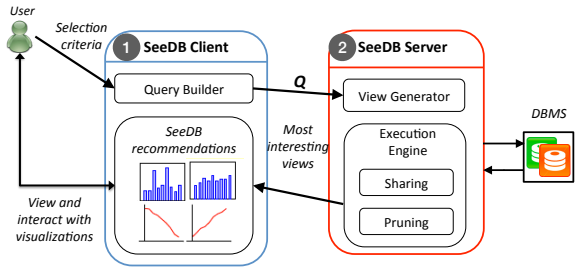


Fig. 13 The architecture of SeeDB [132]. Client (①) accepts users' input, constructs visualization queries and shows recommended visualizations to users. Server (②) generates visualization candidates (view generator) and recommends visualizations to users (execution engine).

Similarity/Distance-based. Zenvisage [118, 119] tries to find other interesting visualizations when the users provide their desired trends, patterns, or insights. Users can draw their desired trends or patterns as a visualization V, then the system recommends visualizations V' by their similarity or dissimilarity (specified by users) with V, i.e., recommends V' with largest or smallest $S(V, V')$, where S is a distance function. Thus the definition of the distance function is of great importance. The distance functions used by Zenvisage are Euclidean distance and Dynamic Time Warping [107]. Figure 12 depicts the frontend of the Zenvisage.

4.2 Behavior-based Recommendations

Behavior-based recommendation systems capture users' current behavior as inputs, then infer users' intended task and recommend useful visualizations based on their tasks.

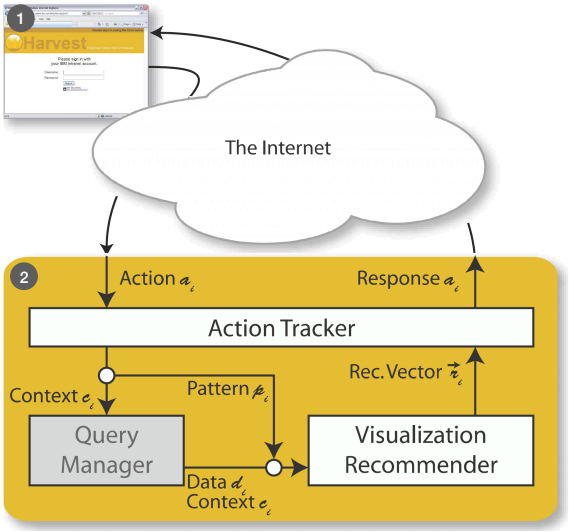


Fig. 14 The architecture of HARVEST [39]. The frontend is a web-based user interface (①). The user can specify the dataset and create data visualizations on the frontend. The backend (②) accepts users' actions, detects patterns, and recommends relevant visualizations by users' task intents, which is inferred by their behavior.

HARVEST [52] is a behavior-driven visualization recommendation system. It recommends visualizations based on the tasks of users which are inferred by their behavior. Since it is difficult for a user to describe her intent clearly and the task of a user evolves as the process of the exploration, HARVEST guesses users' intent by their behavior. As shown in Figure 14, when the user

interacts with HARVEST in the frontend, it captures user's action α_i and sends it to the backend. There are several common atomic actions studied by HARVEST: `inspect`, `filter`, and `bookmark`. The atomic actions form a complex pattern, which usually indicates specific intents. Next, the Action Tracker module (Figure 14–②) analyzes and outputs the user's task context c_i and pattern p_i based on accepted actions. More concretely, c_i denotes the constraints on current data and p_i indicates user's task intents. HARVEST defines 4 patterns: `scan`, `flip`, `swap`, and `drill-down`. Then HARVEST can recommend visualizations by the inferred patterns of users. For example, if a user iteratively `inspect`s hotel price of different regions in a map, HARVEST can detect a `scan` pattern, which means that users want to compare some attributes between some similar objects, thus HARVEST may recommend a bar chart showing the comparison of hotel price of different regions.

4.3 Personalized Recommendations

Personalized recommendation systems capture users' historical behavior as inputs to recommend personalized interesting visualizations.

Linear Model. VizDeck [69] provides personalized visualization recommendation results by training a linear model for each user using their historical behavior. VizDeck provides a new interface design which displays top-k recommended visualizations to users in a grid. The elements displayed in the grid are called vizlets. Users can browse, drop, promote or reorder the vizlets, and the final selected vizlets will be displayed on an interactive dashboard. VizDeck can get users' visualization preference by their historical behavior during the exploration, extract features for these vizlets, and then train a linear model which score vizlets for future recommendation.

Collaborative Filtering. Besides training a model for each user, there are many other techniques [75,112,122] in personalized recommendation systems. For example, collaborative filtering (CF) [112] is a widely used personalized recommendation algorithm. Based on CF, VizRec [91] proposes three methods for personalized visualization recommendation.

1. *Collaborative Filtering.* VizRec constructs an $m \times n$ matrix A, where $A[i][j]$ denotes the rating (e.g., 1 to 7) of the user i on the visualization j in the past. And for a given user u, VizRec first calculates the top-k users who are the most similar (i.e., their ratings for different visualizations are similar.) with u by Pearson correlation coefficient, denoted as $u_1, u_2, ..., u_k$. Then for a visualization candidate v, the rating of u on v is calculated as:
$$r_u^v = \hat{r_u} + \frac{\sum_{i=1,2,...,k} sim(u,u_i)(A[u_i][v] - \hat{r_{u_i}})}{\sum_{i=1,2,...,k} sim(u,u_i)} \quad (2)$$
where $\hat{r_u}$ and $\hat{r_{u_i}}$ denote the average rating of user u and u_i respectively.
2. *Content-based Filtering.* For the users who are new to the system, CF-based recommendation is not applicable. Thus VizRec also develops a content-based recommendation. VizRec defines many features (e.g., attributes of given datasets and mark types) to characterize users and visualizations, and uses the frequency of features to construct user and visualization profiles. VizRec constructs the profile of a user by her current (together historical for old users) annotations of visualizations. And VizRec builds the visualization profile by the aggregation of user profiles for this visualization. Then the user and visualization profiles are transformed to vectors by TF-IDF (Term Frequency-Inverse Document Frequency). And for a given user u and a visualization recommendation candidate v, the similarity of u's and v's vectors can be considered as the ranking score between u and v.
3. *Hybrid Filtering.* A hybrid method of the above two methods will bring a host of benefits (e.g., the algorithm becomes adaptive when the users' interest changed). VizRec uses the weighted sum of the normalized scores of the above two methods as the hybrid filtering score.

4.4 A Summary

Table 7 shows a summary of the supported visualization types, input and ranking metric of the above visualization recommendation systems.

Visualization recommendation systems with empty specification, such as Draco [90], Data2Vis [39] and Rank-by-feature [113], are helpful for users to quickly explore the data when the users are not very familiar with data and desired visualizations. Most of the existing recommendation systems require partial specification, because they permit users' specification for desired visualizations as inputs, e.g., keyword specification in DeepEye [81, 82]. Rule-based solution is in line with person's intuitive understanding of visualizations, but it does not make a complete understanding of human perceptions, just focusing on several interested metrics. Machine learning-based solutions need to collect training data, and the results are hard to interpret, but it may well capture human's cognitive knowledge about

Category		Visualization Recommendation System	Visualization Types				Input	Ranking Metric
			Bar	Pie	Line	Scatter		
Specification-based	Incomplete Specification	Draco [90]	√	√	√	√	–	RankSVM Model
		Data2Vis [39]	√	√	√	√	–	RNN
		APT [83]	√	√	√	√	Keyword	Perceptual Rules
		SAGE [105]	√	√	√	√	Keyword	Perceptual Rules
		Voyager [144, 146]	√	√	√	√	Columns	Perceptual Rules
		Rank-by-feature [113]	√	×	×	√	–	Statistical Rules
		Polaris [125]	√	√	√	√	Columns	Perceptual Rules
		Show Me [84]	√	√	√	√	Columns	Perceptual Rules
		DeepEye [81, 82]	√	√	√	√	Keyword	LambdaMART Algorithm
		Wang et al. [135]	×	×	√	√	Time Series	Perceptual Rules
	Reference-based Specification	SeeDB [132]	√	√	√	√	Query	$S(P[V(D_Q)], P[V(D_R)])$
		Profiler [66]	√	√	√	√	Visualization	$D(VisToCol(v), class)$
		Zenvisage [118, 119]	√	√	√	√	Visualization	$S(V, V')$
Behavior-based		HARVEST [52]	√	√	√	√	Current Behavior	Task Driven
Personalized		VizDeck [69]	√	√	√	√	Historical Voting	Linear Model
		VizRec [91]	√	×	√	√	Historical Rating	CF

Table 7 A summary of visualization recommendation systems, where *Visualization Types* is the supported visualization recommendation types; *Input* is the input of this recommendation system; *Ranking Metric* is the main ranking strategy of this recommendation system.

visualization effectiveness. And the learning model will become smarter when more training data is collected.

Users should specify the reference data or desired pattern in the reference based visualization recommendation, which may be difficult for users who are not familiar with the original data and want to explore the data with the help of the recommendation systems. The advantage is that it is easy to develop such a system, and convenient when users are clear about their needs, e.g., find a line chart with desired trend with Zenvisage [118, 119].

Behavior-based recommendations can recommend visualizations based on inferred tasks, but are limited to the predefined behavior patterns, making it not flexible for users' random behavior.

Personalized recommendations perform differently for different users, because personalized recommendations are customized for different users by their historical behavior.

Besides the above works, there is also a preliminary design of a framework [144] which uses a language, CompassQL, to describe different ranking metrics. CompassQL is a general framework, aiming to describe the ranking metrics of SeeDB [132], Voyager [144–146], VizDeck [69], etc. But there is no implementation of CompassQL yet.

5 Other Research Directions

In this section, we will discuss other research topics that are also relevant to data management issues, but are not yet well studied.

5.1 Data Preparation for Data Visualization

Real-life data is typically dirty [13] and visualizing dirty data may mislead users [128]. This phenomenon has been known for a long time as one type of biased visualizations from the data visualization community. For example, a dataset that is integrated from multiple sources may contain duplicates [43]. Naturally, the data being visualized should be cleaned [80], such as value normalization [37], deduplication [121], missing value imputation and outlier detection. Tableau has integrated Trifacta for data preparation over the entire dataset. The following studies have been conducted, from both database community and visualization community, to investigate the impact of dirty data on data visualization.

– *What-if Analysis for Outliers*: Scorpion [148] allows users to manually pinpoint the outliers from the result of an aggregation query. It then tries to find and remove the predicate that causes such outliers, without affecting the other non-outliers. The problem was formulated as an *influential predicates problem*

and was solved by using techniques from *sensitivity analysis*. As a result, Scorpion can automatically move away outliers identified by users.
- *Evaluating Visualizations with Missing Data*: [123] did a crowdsourced study to measure factors influencing response accuracy, data quality, and confidence in interpretation for time series data with missing values. In particular, it tries a combination of three imputation methods (1) zero-filling, (2) marginal mean, and (3) linear interpolation with four ways of showing the imputed values (i) highlight, (ii) downplay, (iii) annotation, and (iv) information removal. The evaluation over two real-world datasets with 300+ crowd users partially verifies the following hypotheses: (I) Perceived data quality and response accuracy will both degrade as the amount of missing data increases. (II) Highlighting methods will generate higher perceived data quality than downplaying and information removal methods. (III) Linear interpolation will lead to higher perceived confidence and data quality than marginal means or zero-filling as it takes into account local trends in dataset. (IV) Imputed values will lead to higher perceived data quality than removed values.

Research Opportunities.

- *Detecting biased visualizations.* A seemingly good visualization might actually be biased, hence, it requires to detect such visualizations automatically. Many people have approached this problem from a statistics perspective. However, it is also important to study this problem, from the angle of dirty data.
- *Task-aware data cleaning.* Intuitively, it is easier to clean a dataset if the targeting task is known, such as only a small part of data needs to be cleaned, which is cheaper than cleaning the entire dataset in the conventional way. [80] has made a first attempt on this line of work, but apparently more are needed.

5.2 Data Visualization Benchmarks

Like ImageNet or the classic TPC benchmarks, it is important to develop benchmarks for performance and recommendation. The benchmarks should be faithful to the visual analysis tasks, provide reusable traces and data, and in the case of recommendation, have high coverage and quality of its labels. There is an emerging focus on developing benchmarks for performance measures [17, 18, 63].

- A research work, VizNet [60] has presented a large-scale corpus of over 31 million datasets compiled from open data repositories and online visualization galleries. It provides the necessary common baseline for comparing visualization design techniques, and developing benchmark models and algorithms for automating visual analysis.

  Naturally, more needs to be done.

Research Opportunities.

- *Categorization of visualizations.* For ImageNet, it is easy to set categories, such as "balloon" or "strawberry", because the classification task is easier. It is not clear about how to define similar categories for visualizations in a conceptual level, such as "trend" or "distribution".
- *Training data.* Assuming the categories can be provided, there remains a daunting task to label visualizations, and each visualization may have multiple labels. Afterwards, it remains a hard problem on how to use these labeled data, e.g., using which machine learning or deep learning model to predict a good visualization for a given task.

5.3 Data Visualization for Database-related Applications

As mentioned earlier in Section 1, data visualization also plays an important role in database-related applications, such as Excel [7], Google Sheets [4], Oracle Data Visualization Desktop [8], IBM Db2 [6], Amazon Quicksight [1], Microsoft Power BI [9], and many others. Naturally, with the rapid development of visualization techniques, there are more opportunities about using data visualization for database-related applications.

Research Opportunities.

- *Data visualization for data discovery.* Data discovery, the problem of finding interesting datasets for a certain application from a data lake with thousands or millions of data silos, remains a hard problem to solve [36, 47]. One roadblock is to quickly understand the discovered datasets. Practically, browsing each dataset is time-consuming. Intuitively, data visualization that provides a high level understanding can help in this important problem.
- *Data visualization for data debugging.* One problem that was recently raised by the Data Civilizer system [103, 104] is data debugging, where the output of a data analytics workflow is wrong not because of bugs in programs, but in the data such as erroneous input or wrong parameters. Although [104] has some initial attempt to combine data visualization for data debugging, the solution for data debugging is far from being mature, and evidently, data

visualization can help for more effective data debugging.

6 Conclusion

Data visualization is a fast growing field with a great many new research results and novel systems developed recently. Research and practitioners from many fields have contributed to the remarkable success of data visualization, which is driven by most (if not all) domains and applications.

This article mainly surveys recent data visualization works, from data management perspective. In particular, we have comprehensively described the works in visualization specifications, efficient methods for data visualization, and visualization recommendation. As mentioned earlier, most commercial data visualization systems are good at ease-of-use in terms of data visualization specifications. However, many practitioners are still suffering from the efficiency and recommendation issues of these systems. Hence, we also discuss several open problems that database researchers can make significant contribution to advance the field of data visualization.

Acknowledgement. This work was supported by 973 Program of China (2015CB358700), NSF of China (61632016, 61472198, 61521002, 61661166012), Huawei, and TAL education.

References

1. Amazon quicksight: Cloud based business intelligence. `https://aws.amazon.com/quicksight/`.
2. Echarts. `http://echarts.baidu.com`.
3. Flare. `http://flare.prefuse.org`.
4. Google sheets: Free online spreadsheets for personal use. `https://www.google.com/sheets/about/`.
5. Hyper: A hybrid oltp and olap high performance dbms. `https://hyper-db.de`.
6. Ibm db2. `https://www.ibm.com/analytics/db2`.
7. Microsoft excel. `https://products.office.com/en-us/excel`.
8. Oracle data visualization desktop. `https://docs.oracle.com/en/middleware/bi/data-visualization-desktop/tutorials.html`.
9. Power bi: Interactive data visualization bi tools. `https://powerbi.microsoft.com`.
10. Qlik: Data analytics for modern business intelligence. `https://www.qlik.com/us`.
11. Tableau. `https://www.tableau.com`.
12. Vega: A visualization grammar. `https://vega.github.io/vega/`.
13. Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.
14. G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, (6):734–749, 2005.
15. D. Alabi and E. Wu. Pfunk-h: approximate query processing using perceptual models. In *HILDA@SIGMOD*, pages 10–16, 2016.
16. D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *Computer Science*, 2014.
17. L. Battle, M. Angelini, C. Binnig, T. Catarci, P. Eichmann, J. Fekete, G. Santucci, M. Sedlmair, and W. Willett. Evaluating visual data analysis systems: A discussion report. In *HILDA@SIGMOD*, pages 4:1–4:6, 2018.
18. L. Battle, R. Chang, J. Heer, and M. Stonebraker. Position statement: The case for a visualization performance benchmark. In *DSIA*, pages 1–5, 2017.
19. L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *SIGMOD*, pages 1363–1375, 2016.
20. R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. In *Software pioneers*, pages 245–262. Springer, 2002.
21. F. Beck, M. Burch, S. Diehl, and D. Weiskopf. A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum*, 36(1):133–159, 2017.
22. J. Bertin. Semiology of graphics. 1983.
23. N. Bikakis. Big data visualization tools. 2018.
24. N. Bikakis, G. Papastefanatos, M. Skourla, and T. Sellis. A hierarchical aggregation framework for efficient multilevel visual exploration and analysis. *Semantic Web*, 8(1):139–179, 2017.
25. N. Bikakis and T. Sellis. Exploration and visualization in the web of big linked data: A survey of the state of the art. *arXiv preprint arXiv:1601.08059*, 2016.
26. J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.
27. M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *TVCG*, 15(6):1121–8, 2009.
28. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.
29. R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.
30. S. M. Casner. Task-analytic approach to the automated design of graphic presentations. *Acm Transactions on Graphics*, 10(2):111–151, 1991.
31. S.-M. Chan, L. Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *VAST*, pages 59–66, 2008.
32. J. R. Christi and K. Premkumar. Survey on recommendation and visualization techniques for qos-aware web services. In *ICICES*, pages 1–6, 2014.
33. W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *ASA*, 74(368):829–836, 1979.
34. W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *ASA*, 79(387):531–554, 1984.
35. A.-S. Dadzie and E. Pietriga. Visualisation of linked data–reprise. *Semantic Web*, 8(1):1–21, 2017.
36. D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *CIDR*, 2017.

37. D. Deng, W. Tao, Z. Abedjan, A. K. Elmagarmid, I. F. Ilyas, G. Li, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Unsupervised string transformation learning for entity consolidation. In *ICDE*, pages 196–207, 2019.
38. M. Diamond and A. Mattia. Data visualization: An exploratory study into the software tools used by businesses. *Journal of Instructional Pedagogies*, 18, 2017.
39. V. Dibia and Ç. Demiralp. Data2vis: Automatic generation of data visualizations using sequence to sequence recurrent neural networks. *CoRR*, abs/1804.03126, 2018.
40. B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample + seek: Approximating aggregates with distribution precision guarantee. In *SIGMOD*, pages 679–694, 2016.
41. H. L. Dong, J. S. Kim, S. D. Kim, K. C. Kim, K. Yoosung, and J. Park. Adaptation of a neighbor selection markov chain for prefetching tiled web gis data. In *ADVIS*, pages 213–222, 2002.
42. P. R. Doshi, E. A. Rundensteiner, and M. O. Ward. Prefetching for visual data exploration. In *DASFAA*, pages 195–202, 2003.
43. M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *PVLDB*, 11(11):1454–1467, 2018.
44. N. Elmqvist and J. D. Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *TVCG*, 16(3):439–454, 2010.
45. S. V. D. Elzen and J. J. van Wijk. Small multiples, large singles: A new approach for visual data exploration. *Computer Graphics Forum*, 32(3pt2):191–200, 2013.
46. T. Epelbaum. Deep learning: Technical introduction. *CoRR*, abs/1709.01412, 2017.
47. R. C. Fernandez, E. Mansour, A. A. Qahtan, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE*, pages 989–1000, 2018.
48. D. Fisher, I. Popov, S. Drucker, and m. schraefel. Trust me, i'm partially right: Incremental visualization lets analysts explore large datasets faster. In *CHI*, pages 1673–1682, 2012.
49. Y. H. Fua, M. O. Ward, and E. A. Rundensteiner. Structure-based brushes: a mechanism for navigating hierarchically organized data and information spaces. *TVCG*, 6(2):150–159, 2000.
50. A. Ghosh, M. Nashaat, J. Miller, S. Quader, and C. Marston. A comprehensive review of tools for exploratory analysis of tabular industrial datasets. *Visual Informatics*, 2(4):235–253, 2018.
51. H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, and J. Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *SIGMOD*, pages 1061–1066, 2010.
52. D. Gotz and Z. Wen. Behavior-driven visualization recommendation. In *IUI*, pages 315–324, 2009.
53. I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yogev, and S. Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *RecSys*, pages 53–60, 2009.
54. P. Hanrahan. Vizql: a language for query, analysis and visualization. In *SIGMOD*, page 721, 2006.
55. J. Heer, S. K. Card, and J. A. Landay. prefuse: a toolkit for interactive information visualization. In *CHI*, pages 421–430, 2005.
56. R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *ICANN*, pages 97–102 vol.1, 2002.
57. I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *TVCG*, 6(1):24–43, 2000.
58. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
59. E. Hoque, V. Setlur, M. Tory, and I. Dykeman. Applying pragmatics principles for interaction with visual analytics. *TVCG*, 24(1):309–318, 2017.
60. K. Z. Hu, S. N. S. Gaikwad, M. Hulsebos, M. A. Bakker, E. Zgraggen, C. A. Hidalgo, T. Kraska, G. Li, A. Satyanarayan, and Ç. Demiralp. Viznet: Towards A large-scale visualization learning and benchmarking repository. In *CHI*, page 662, 2019.
61. K. Z. Hu, D. Orghian, and C. A. Hidalgo. DIVE: A mixed-initiative system supporting integrated data exploration workflows. In *HILDA@SIGMOD*, pages 5:1–5:7, 2018.
62. S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *SIGMOD*, pages 277–281, 2015.
63. L. Jiang, P. Rahman, and A. Nandi. Evaluating interactive data systems: Workloads, metrics, and guidelines. In *SIGMOD*, pages 1637–1644, 2018.
64. A. Kalinin, U. Cetintemel, and S. Zdonik. Interactive data exploration using semantic windows. In *SIGMOD*, pages 505–516, 2014.
65. S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.
66. S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *AVI*, pages 547–554, 2012.
67. A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. Ontology visualization methods—a survey. *ACM Computing Surveys (CSUR)*, 39(4):10, 2007.
68. D. A. Keim, J. P. Lee, B. Thuraisinghaman, and C. Wittenbrink. Database issues for data visualization: supporting interactive database exploration. In *Workshop on Database Issues for Data Visualization*, pages 12–25, 1995.
69. A. Key, B. Howe, D. Perry, and C. R. Aragon. Vizdeck: self-organizing dashboards for visual analytics. In *SIGMOD*, pages 681–684, 2012.
70. M. Khan and S. S. Khan. Data and information visualization methods, and interactive mechanisms: A survey. *International Journal of Computer Applications*, 34(1):1–14, 2011.
71. A. Kim, E. Blais, A. G. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 8(5):521–532, 2015.
72. Y. Kim and J. Heer. Assessing effects of task and data distribution on the effectiveness of visual encodings. *Computer Graphics Forum*, 37(3):157–167, 2018.
73. D. Li, H. Mei, Y. Shen, S. Su, W. Zhang, J. Wang, M. Zu, and W. Chen. Echarts: A declarative framework for rapid construction of web-based visualization. *Visual Informatics*, 2018.
74. L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *TVCG*, 19(12):2456–2465, 2013.

75. R. R. Liu, C. X. Jia, T. Zhou, D. Sun, and B. H. Wang. Personal recommendation via modified collaborative filtering. *Physica A Statistical Mechanics and Its Applications*, 388(4):462–468, 2012.
76. S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. *TVCG*, (3):1249–1268, 2017.
77. Z. Liu, B. Jiang, and J. Heer. immens : real-time visual querying of big data. In *Eurographics Conference on Visualization*, pages 421–430, 2013.
78. Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data illustrator. `http://www.zcliu.org/di/`.
79. Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. T. Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *CHI*, page 123, 2018.
80. Y. Luo, C. Chai, X. Qin, N. Tang, and G. Li. Interactive cleaning for progressive visualization through composite questions. In *ICDE*, 2020.
81. Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: Towards automatic data visualization. In *ICDE*, pages 101–112, 2018.
82. Y. Luo, X. Qin, N. Tang, G. Li, and X. Wang. Deepeye: Creating good data visualizations by keyword search. In *SIGMOD*, pages 1733–1736, 2018.
83. J. Mackinlay. Automating the design of graphical presentations of relational information. *Acm Transactions On Graphics*, 5(2):110–141, 1986.
84. J. D. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *TVCG*, 13(6):1137–1144, 2007.
85. N. Marie and F. Gandon. Survey of linked data based exploration systems. In *IESD*, 2014.
86. H. Mei, Y. Ma, Y. Wei, and W. Chen. The design space of construction tools for information visualization: A survey. *Journal of Visual Languages & Computing*, 44:120–132, 2018.
87. B. Michael, O. Vadim, and H. Jeffrey. D3: Data-driven documents. *TVCG*, 17(12):2301–9, 2011.
88. D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. In *CHI*, pages 2904–2915, 2017.
89. D. Moritz, B. Howe, and J. Heer. Falcon: Balancing interactive latency and resolution sensitivity for scalable linked visualizations. 2019.
90. D. Moritz, C. Wang, G. L. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *TVCG*, 25(1):438–448, 2019.
91. B. Mutlu, E. Veas, and C. Trattner. Vizrec: Recommending personalized visualizations. *TIIS*, 6(4):31, 2016.
92. T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.
93. T. Neumann, T. Mühlbauer, and A. Kemper. Fast serializable multi-version concurrency control for main-memory database systems. In *SIGMOD*, pages 677–689, 2015.
94. C. A. Pahins, S. A. Stephens, C. Scheidegger, and J. L. Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *TVCG*, 23(1):671–680, 2016.
95. Z. Pang, S. Wu, G. Chen, K. Chen, and L. Shou. Flashview: An interactive visual explorer for raw data. *PVLDB*, 10(12):1869–1872, 2017.
96. H. Piringer, C. Tominski, P. Muigg, and W. Berger. A multi-threading architecture to support interactive visual exploration. *TVCG*, 15(6):1113–1120, 2009.
97. J. Poco and J. Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. In *Computer Graphics Forum*, pages 353–363, 2017.
98. X. Qin, Y. Luo, N. Tang, and G. Li. Deepeye: An automatic big data visualization framework. *Big Data Mining and Analytics*, 1(1):75–82, 2018.
99. X. Qin, Y. Luo, N. Tang, and G. Li. Deepeye: Visualizing your data by keyword search. *EDBT Vision*, 2018.
100. J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
101. S. Rahman, M. Aliakbarpour, H. K. Kong, E. Blais, K. Karahalios, A. Parameswaran, R. Rubinfield, S. Rahman, M. Aliakbarpour, and H. K. Kong. I've seen "enough": incrementally improving visualizations to support rapid decision making. *PVLDB*, 10(11):1262–1273, 2017.
102. D. Ren, T. Höllerer, and X. Yuan. ivisdesigner: Expressive interactive design of information visualizations. *TVCG*, 20(12):2092–2101, 2014.
103. E. K. Rezig, L. Cao, G. Simonini, M. Schoemans, S. Madden, M. Ouzzani, N. Tang, and M. Stonebraker. Dagger: A data (not code) debugger. In *CIDR*, 2020.
104. E. K. Rezig, L. Cao, M. Stonebraker, G. Simonini, W. Tao, S. Madden, M. Ouzzani, N. Tang, and A. K. Elmagarmid. Data civilizer 2.0: A holistic framework for data preparation and analytics. *PVLDB*, 12(12):1954–1957, 2019.
105. S. F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein. Interactive graphic design using automatic presentation knowledge. In *CHI*, page 207, 1994.
106. B. Saket, A. Endert, and C. Demiralp. Task-based effectiveness of basic visualizations. *TVCG*, PP(99):1–1, 2017.
107. H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):159–165, 1990.
108. A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. `https://idl.cs.washington.edu/projects/lyra/`.
109. A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *TVCG*, 23(1):341–350, 2016.
110. A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *TVCG*, 22(1):659–668, 2015.
111. G. Scagnostics. *Graph-Theoretic Scagnostics*. 2005.
112. J. B. Schafer, J. Konstan, and J. Riedl. Recommender systems in e-commerce. In *World Automation Congress*, pages 158–166, 1999.
113. J. Seo and B. Shneiderman. A rank-by-feature framework for interactive exploration of multidimensional data. *IV*, 4(2):96–113, 2005.
114. V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang. Eviza: A natural language interface for visual analysis. In *UIST*, pages 365–377, 2016.
115. L. Sharma and A. Gera. A survey of recommendation system: Research challenges. *IJETT*, 4(5):1989–1992, 2013.
116. R. N. Shepard. Toward a universal law of generalization for psychological science. *Science*, 242(4880):1317–1323, 1988.

117. B. Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69, 1983.
118. T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. G. Parameswaran. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.
119. T. Siddiqui, J. Lee, A. Kim, E. Xue, X. Yu, S. Zou, L. Guo, C. Liu, C. Wang, K. Karahalios, and A. G. Parameswaran. Fast-forwarding to desired visualizations with zenvisage. In *CIDR*, 2017.
120. B. W. Silverman. Density estimation for statistics and data analysis. 2018.
121. R. Singh, V. V. Meduri, A. K. Elmagarmid, S. Madden, P. Papotti, J. Quiané-Ruiz, A. Solar-Lezama, and N. Tang. Synthesizing entity matching rules by examples. *PVLDB*, 11(2):189–202, 2017.
122. I. Soboroff and C. Nicholas. Combining content and collaboration in text filtering. In *IJCAI*, pages 86–91, 1999.
123. H. Song and D. A. Szafir. Where's my data? evaluating visualizations with missing data. *IEEE Trans. Vis. Comput. Graph.*, 25(1):914–924, 2019.
124. StephanLewandowsky and IanSpence. Discriminating strata in scatterplots. *ASA*, 84(407):682–688, 1989.
125. C. Stolte and P. Hanrahan. Polaris: A system for query, analysis and visualization of multi-dimensional relational databases. In *INFOVIS*, pages 5–14, 2000.
126. M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *Interspeech*, pages 601–608, 2012.
127. I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, volume 4, pages 3104–3112, 2014.
128. N. Tang, E. Wu, and G. Li. Towards democratizing relational data visualization. In *SIGMOD*, pages 2025–2030, 2019.
129. W. Tao, X. Liu, Ç. Demiralp, R. Chang, and M. Stonebraker. Kyrix: Interactive visual data exploration at scale. In *CIDR*, 2019.
130. F. Tauheed, T. Heinis, F. Shürmann, H. Markram, and A. Ailamaki. SCOUT: Prefetching for latent feature following queries. *PVLDB*, 5(11):1531–1542, 2012.
131. VanderPlas, Jacob, B. E. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert. Altair: Interactive statistical visualizations for python. `https://altair-viz.github.io`.
132. M. Vartak, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: automatically generating query visualizations. *PVLDB*, 7(13):1581–1584, 2014.
133. M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.
134. T. Von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner. Visual analysis of large graphs: state-of-the-art and future research challenges. In *Computer Graphics Forum*, volume 30, pages 1719–1749, 2011.
135. Y. Wang, F. Han, L. Zhu, O. Deussen, and B. Chen. Line graph or scatter plot? automatic selection of methods for visualizing trends in time series. *TVCG*, 24(2):1141–1154, 2018.
136. Z. Wang, N. Ferreira, Y. Wei, A. S. Bhaskar, and C. Scheidegger. Gaussian cubes: Real-time modeling for visual exploration of large multidimensional datasets. *TVCG*, 23(1):681–690, 2016.
137. L. Warren. The visual display of quantitative information. *Yale Journal of Biology and Medicine*, 44(4):400–400, 1986.
138. K. Wei, J. Huang, and S. Fu. A survey of e-commerce recommender systems. In *ICSSSM*, pages 1–5, 2007.
139. R. M. G. Wesley, M. Eldridge, and P. Terlecki. An analytic data engine for visualization in tableau. In *SIGMOD*, pages 1185–1194, 2011.
140. H. Wickham. ggplot2 - elegant graphics for data analysis. 2009.
141. H. Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.
142. L. Wilkinson. *The grammar of graphics*. Springer, 2005.
143. E. Wohlfart, W. Aigner, A. Bertone, and S. Miksch. Comparing information visualization tools focusing on the temporal dimensions. In *IV*, pages 69–74, 2008.
144. K. Wongsuphasawat, D. Moritz, A. Anand, J. D. Mackinlay, B. Howe, and J. Heer. Towards a general-purpose query language for visualization recommendation. In *HILDA@SIGMOD*, pages 4–9, 2016.
145. K. Wongsuphasawat, D. Moritz, A. Anand, J. D. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *TVCG*, 22(1):649–658, 2016.
146. K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. D. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *CHI*, pages 2648–2659, 2017.
147. E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: vision paper. *PVLDB*, 7(10):903–906, 2014.
148. E. Wu and S. R. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, pages 553–564, 2013.
149. E. Wu, F. Psallidas, Z. Miao, H. Zhang, and L. Rettig. Combining design and performance in a data visualization management system. In *CIDR*, 2017.
150. E. Wu, F. Psallidas, Z. Miao, H. Zhang, L. Rettig, Y. Wu, and T. Sellam. Combining design and performance in a data visualization management system. In *CIDR*, 2017.
151. Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Ranking, boosting, and model adaptation. Technical report, Technical report, Microsoft Research, 2008.
152. M. A. Yalçın, N. Elmqvist, and B. B. Bederson. Keshif: Rapid and expressive tabular data exploration for novices. *TVCG*, 24(8):2339–2352, 2018.
153. S. Yesilmurat. Retrospective adaptive prefetching for interactive web gis applications. *Geoinformatica*, 16(3):435–466, 2012.
154. K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *SIGMOD*, pages 1555–1566, 2014.